Fusion

Version 5.4

Lucidworks | Make Search a Superpower™

Table of Contents

- *About Legacy Docs*
- *Introduction to Fusion*
  - *Fusion UI Tour*
    - *Fusion Apps*
    - *Index Workbench*
    - *Query Workbench*
    - *Object Explorer*
  - *The Fusion Stack*
    - *Apache Solr*
    - *Milvus*
    - *Spark Operations*
      - *Spark Job Drivers*
    - *ZooKeeper*
  - *Kubernetes Deployment Architecture*
  - *Indexing Data Flow*
  - *Query Data Flow*
  - *Machine Learning*
    - *Machine Learning Jobs*
    - *Machine Learning Models in Fusion*
    - *Natural Language Processing*
- *Getting Data In*
  - *Indexing Your Data*
    - *Indexing Data Flow*
    - *Collections*
      - *Fields Editor UI*
    - *The Indexing Workbench*
    - *Index Pipelines*
      - *Index Pipeline Stages*
      - *Index Profiles*
      - *Custom JavaScript Stages for Index Pipelines*
    - *Indexing Signals*
    - *Data Models*
    - *Subscriptions UI*
      - *Apache Pulsar*
    - *Connectors and Datasources*
  - *Blob Storage*
  - *Importing Signals*
  - *Parallel Bulk Loader (PBL)*
- *Getting Data Out*
  - *Query Basics*
    - *Fusion SQL*
    - *Query Data Flow*
    - *Query Pipelines*
      - *Query Pipeline Stages*
      - *Query Profiles*
      - *Custom JavaScript Stages for Query Pipelines*
    - *The Query Workbench*
    - *Solr Query Language Cheat Sheet*
    - *Domain Specific Language (DSL)*
    - *Faceting*
    - *Stored Parameters Collection*
  - *Query Enhancement*
    - *Signals*
      - *Signals Data Flow*
      - *Aggregations*
      - *Signals Types and Structures*
      - *Deleting Old Signals*
      - *Required Signals for App Insights*
      - *Required Signals for Jobs*
    - *Boosting*
      - *Boost With Signals*
    - *Recommendations*
      - *Getting Started with Recommendations and Boosting*
      - *Methods for Recommendations and Boosting*

        - *Items-For-Item Recommendations*
        - *Items-For-Query Recommendations*
        - *Items-For-User Recommendations*
        - *Queries-for-Query Recommendations*
        - *Trending Items and Queries*
        - *Users-for-Item Recommendations*
    - *Synonyms*
        - *Synonyms Files*
    - *Business Rules*
        - *General Configurations*
        - *Conditions*
        - *Actions*
        - *Custom Rule Actions*
    - *Head/Tail Analysis*
    - *Template Expressions*
        - *Index Pipeline Stage Template Expression Variables*
        - *Messaging Services Template Expression Variables*
        - *Query Pipeline Stage Template Expression Variables*
    - *Misspelling Detection*
    - *Phrase Detection*
    - *Query Rewriting*
    - *Remove Words*
    - *Response Rewriting*
    - *Rules Editor*
    - *Stopwords Files*
- *Curating Your Search Experience*
    - *Predictive Merchandiser*
        - *Hierarchical Facets*
        - *Licensing for Predictive Merchandiser*
        - *Merchandising Zero Results*
        - *Product Grouping*
        - *Rule Conflict Resolution*
        - *Search Rewrites*
        - *Supported Browsers*
        - *Templates*
        - *Templates UI Configuration Options*
    - *Experience Optimizer*
        - *Document Grouping*
        - *Licensing for Experience Optimizer*
        - *Optimizing Zero Results*
        - *Rule Conflict Resolution*
        - *Search Rewrites*
        - *Supported Browsers*
- *Data Analytics*
    - *Querying Fusion SQL*
        - *2D K-Means Clustering (kmeans)*
        - *Co-occurrence Matrices (co_matrix)*
        - *Correlation Matrices (corr_matrix)*
        - *Cumulative Probability (cumulative_probability)*
        - *Fusion SQL Aggregation*
        - *Fusion SQL Miscellaneous Functions*
        - *Fusion SQL Search*
        - *Fusion SQL Statistics*
        - *Fusion SQL Text Analytics*
        - *Gaussfit (gaussfit)*
        - *Geo Clustering (geo_cluster)*
        - *Histogram (hist)*
        - *Knnfit (knnfit)*
        - *Mann-Whitney (mann_whitney)*
        - *Percentiles (percentiles)*
        - *Polyfit (polyfit)*
        - *Probability (probability)*
        - *QQPlot*
        - *Regression Diagnostics (regress)*
        - *Regression and Residual Plot (linearfit)*
        - *Significant Terms (significant_terms)*
        - *Text Analysis and Analytics (analyze)*
        - *Time Series (date_format)*

        - *click_pos*
        - *concat_text*
        - *timediff*
        - *ttest*
        - *weighted_sum*
      - *App Insights*
        - *Dashboards in App Insights*
        - *Events*
        - *Sessions*
        - *Analytics*
        - *Experiment Results*
        - *Requirements for Reports and Dashboards*
      - *Experiments*
        - *Experiment Metrics*
      - *Jupyter Support in Fusion*
    - *Advanced Query Enhancement*
      - *Smart Answers*
        - *Best Practices*
        - *Cold Start Solution*
        - *FAQ Solution*
        - *Advanced Model Training Configuration for Smart Answers*
      - *Sentiment Analysis and Prediction*
- *Operations*
  - *Fusion 5 Survival Guide*
    - *Which Kubernetes?*
    - *Multiple Zones for High Availability*
    - *Ingress and Security*
    - *Stateless Sessions with JWT*
    - *High-Performance Query Processing with Auto-Scaling*
    - *Workload Isolation with Multiple Node Pools*
    - *GPU Support for Deep Learning Training*
    - *Fusion 5 On-prem Kubernetes Deployment*
    - *Spark Administration in Kubernetes*
  - *Kubernetes Deployment*
  - *Jobs and Scheduling*
    - *Fusion Schedules*
    - *Datasource Jobs*
    - *Job History*
    - *Spark Jobs*
    - *Tasks*
  - *Security*
    - *Access Control*
      - *Permissions*
      - *Roles*
      - *Security Realms*
      - *Users*
    - *High Availability and Disaster Recovery*
    - *Ingress and Security*
  - *Monitoring*
    - *Grafana Dashboards*
    - *Banana Dashboards*
      - *Built-in Dashboards*
      - *Display Panels*
      - *Input Panels*
    - *Prometheus and Grafana in Fusion*
    - *The DevOps Center*
  - *Fusion 5 Frequently Asked Questions*
- *Reference*
  - *Configuration Reference*
    - *Jobs*
      - *ALS Recommender Jobs*
        - *Items-for-Item Recommendations Configuration (ALS)*
        - *Items-for-User Recommendations Configuration (ALS)*
      - *Aggregation Jobs*
        - *Aggregator Functions Examples*
        - *Built-in SQL Aggregation Jobs*
        - *SQL Aggregation Examples*
        - *SQL Aggregation Jobs*

- *BPR Recommender Jobs*
- *Build Training Data Jobs*
- *Classification Jobs*
- *Cluster Labeling Jobs*
- *Content-Based Recommender Jobs (Experimental)*
- *Create Collections in Milvus Jobs*
- *Create Indexes in Milvus Jobs*
- *Create Seldon Core Model Deployment*
- *Custom Python Jobs*
- *Custom Spark Jobs*
- *Data Augmentation Job*
- *Delete Collections in Milvus Jobs*
- *Delete Indexes in Milvus Jobs*
- *Delete Seldon Core Model Deployment Job*
- *Document Clustering Jobs*
- *Download Blob*
- *Ground Truth Jobs*
- *Head/Tail Analysis Jobs*
- *Log Cleanup Jobs*
- *Outlier Detection Jobs*
- *Parallel Bulk Loader*
- *Phrase Extraction Jobs*
- *Query-to-Query Collaborative Similarity Job*
- *Query-to-Query Session-Based Similarity Jobs*
- *REST Call Jobs*
- *Ranking Metrics Jobs*
- *Recreate a Milvus Collection*
- *SQL-Based Experiment Metric Jobs*
- *Script Jobs*
- *Smart Answers Coldstart Training*
- *Smart Answers Evaluate Pipeline*
- *Smart Answers Supervised Training*
- *Synonym Detection Jobs*
- *Token and Phrase Spell Correction Job*
- *Transfer Collection to Cloud Job*
- *Trending Recommender Jobs*
- *Upload Model Parameters to Cloud Job*

- *Pipeline Stages*
  - *Query Pipeline Stages Reference*
    - *Active Directory Security Trimming Stage*
    - *Additional Query Parameters Stage*
    - *Analytics Catalog Stage*
    - *Apply Rules Stage*
    - *Block Documents Stage*
    - *Boost Documents Stage*
    - *Boost with Signals Stage*
    - *DSL to Legacy Parameters Stage*
    - *Experiment Stage*
    - *Field Facet Stage*
    - *Highlight Stage*
    - *JDBC Lookup Stage*
    - *JavaScript Query Stage*
    - *Landing Pages Stage*
    - *Logging Stage*
    - *Machine Learning Stage*
    - *Managed JavaScript Query Stage*
    - *Merge Async Results Stage*
    - *Milvus Ensemble Query Stage*
    - *Milvus Query Stage*
    - *Milvus Response Update Stage*
    - *Modify Response with Rules Stage*
    - *NLP Annotator Query Stage*
    - *Parameterized Boosting Stage*
    - *Query Fields Stage*
    - *REST Query Stage*
    - *Recommend Items for Item Stage*
    - *Recommend Items for User Stage*
    - *Recommend Similar Queries Stage*

        - *Response Document Exclusion Stage*
        - *Response Pairwise Swap Stage*
        - *Response Shuffle Stage*
        - *Return Query Parameters Stage*
        - *Rollup Aggregation Stage*
        - *Run Query Pipeline Stage*
        - *Security Trimming Stage*
        - *Solr MoreLikeThis Stage*
        - *Solr Query Stage*
        - *Solr Subquery Stage*
        - *Text Tagger Stage*
      - *Index Pipeline Stages Reference*
        - *Call Data Model Pipeline Index Stage*
        - *Call Pipeline Index Stage*
        - *Data Model Mapping Index Stage*
        - *Date Parsing Index Stage*
        - *Detect Language Index Stage*
        - *Encode into Milvus Index Stage*
        - *Exclude Documents Index Stage*
        - *Exclusion Filter Index Stage*
        - *Field Mapping Index Stage*
        - *Field Parser Index Stage*
        - *Filter Short Fields Index Stage*
        - *Find and Replace Index Stage*
        - *Format Signals Index Stage*
        - *Gazetteer Lookup Extraction Index Stage*
        - *GeoIP Lookup Stage*
        - *Include Documents Index Stage*
        - *JDBC Index Stage*
        - *JavaScript Index Stage*
        - *Logging Index Stage*
        - *Machine Learning Index Stage*
        - *Managed JavaScript Index Stage*
        - *NLP Annotator Index Stage*
        - *OpenNLP NER Extraction Index Stage*
        - *REST Query Index Stage*
        - *Regex Field Extraction Index Stage*
        - *Regex Field Filter Index Stage*
        - *Regex Field Replacement Index Stage*
        - *Resolve Multivalued Fields Index Stage*
        - *Send to Message Bus Index Stage*
        - *Set Property Index Stage*
        - *Solr Dynamic Field Name Mapping*
        - *Solr Indexer Stage*
        - *Solr Partial Update Indexer Stage*
        - *Text Processing Index Stage*
        - *Update Experiment Stage*
        - *Update Related Document Stage*
        - *XML Transformation Index Stage*
      - *Custom JavaScript Stages Global Variables*
    - *Parsers*
      - *Apache Tika Parser*
      - *Apache Tika Server Stage*
      - *Archive Parser Stage*
      - *CSV Parser Stage*
      - *Fallback Parser Stage*
      - *Forked Apache Tika Parser Stage*
      - *Grok Parser Stage*
      - *Grok Patterns*
      - *HTML Parser Stage*
      - *JSON Parser Stage*
      - *Solr Update Parser Stage*
      - *Text Parser Stage*
      - *XML Parser Stage*
    - *Configure Fusion for JWT*
  - *Fusion Microservices*
  - *Custom JavaScript Query Stage examples*
  - *Docker Images by Fusion Version*

    - *Gateway Configuration Properties*
    - *Language Support in Fusion*
    - *Solr Reference Guide*
    - *Spark Reference Guide*
- *Developer Docs*
    - *Fusion REST APIs*
        - *Admin APIs*
            - *Apps API*
            - *Blob Store API*
            - *Collection Features API*
            - *Collections API*
            - *Configurations API*
            - *Data Models API*
            - *History API*
            - *Jobs API*
            - *Links API*
            - *Nodes API*
            - *Objects API*
            - *Scheduler API*
            - *Search Cluster API*
            - *Solr API*
            - *Solr Configuration API*
            - *SolrAdmin API*
            - *System API*
            - *ZooKeeper Import/Export API*
        - *Query Service APIs*
            - *Query Rewrite API Configurations*
                - *Query Rewrite API*
                - *Banner Rule API Configurations*
                - *Block List Rule API Configurations*
                - *Boost Attributes Rule API Configurations*
                - *Boost List Rule API Configurations*
                - *Bury List Rule API Configurations*
                - *Filter List Rule API Configurations*
                - *Ingroup Boost List Rule API Configurations*
                - *Ingroup Bury List Rule API Configurations*
                - *Ingroup Pinned List Rule API Configurations*
                - *JSON Blob Rule API Configurations*
                - *Pinned Rule API Configurations*
                - *Redirect Rule API Configurations*
                - *Response Value Rule API Configurations*
                - *Set Facets Rule API Configurations*
                - *Set Params Rule API Configurations*
                - *Ground Truth Query Rewrite API Configurations*
                - *Phrase Query Rewrite API Configurations*
                - *Remove Words Query Rewrite API Configurations*
                - *Similar Queries (simq) Query Rewrite API Configurations*
                - *Spell Query Rewrite API Configurations*
                - *Synonym Query Rewrite API Configurations*
                - *Tail Query Rewrite API Configurations*
            - *Custom Rules API*
            - *Experiments API*
            - *Query API*
            - *Query Pipelines API*
            - *Query Profiles API*
            - *Query Stage Plugins API*
            - *Query Stages API*
        - *Connectors Service APIs*
            - *Connector Datasources API*
            - *Connector Plugins API*
            - *Connector Plugins Repository API*
        - *Fusion Proxy APIs*
            - *Realms API*
            - *Roles API*
            - *Sessions API*
            - *User API*
        - *Indexing Service APIs*
            - *Index Pipelines API*

        - *Index Profiles API*
        - *Index Stages API*
        - *Parsers API*
    - *Job Launcher APIs*
    - *Job Rest Server APIs*
        - *Recommendations API*
        - *Signals API*
        - *Spark Jobs API*
    - *Machine Learning Model Service APIs*
        - *ML Model API*
        - *Serialized Model API*
    - *SQL Service APIs*
    - *Templating API*
- *Front-end Development*
    - *Autocomplete*
    - *REST API Examples for Front-End Development*
- *Connectors SDK*
    - *Java Connector Development*
    - *Simple Demo Connector*
    - *Connectors SDK Javadocs*
- *Index Stage SDK*
    - *Index Stage SDK Javadocs*
- *Pipeline Javadocs*
- *Fusion App Studio*
    - *App Studio Reference Guides*
        - *Services*
            - *$filterService*
            - *$lightningLog*
            - *$lightningUrl*
            - *$twigkit*
            - *$util*
            - *BookmarkService*
            - *CollaborationService*
            - *DateUtils*
            - *FieldService*
            - *ModalService*
            - *PlatformService*
            - *QueryIdService*
            - *QueryService*
            - *ResponseService*
            - *ScrollToService*
            - *TogglePanesService*
            - *TranslationService*
            - *UserService*
            - *httpInterceptor*
            - *scrollAndResizeListener*
        - *Accessing Data*
            - *Configuring the Search Platform*
            - *Filter Syntax*
            - *Formulating a Query*
            - *Search Responses*
        - *Modules*
            - *Activity Tracking*
            - *Visualizations*
                - *Adding Credits*
                - *Chart Legend*
                - *Chart Tooltips*
                - *Displaying a Chart*
                - *Exporting a Chart*
                - *Faceting a Data Series*
                - *Histogram*
                - *Search Result Data Series*
                - *Tag Cloud*
            - *Security*
                - *Authentication*
                    - *OAuth*
                    - *SAML Configuration*
                        - *Troubleshooting SAML*

        - *Role-based Access*
        - *User Web Service*
    - *Geospatial Module*
        - *Esri Web Maps*
        - *Maps*
    - *Media Module*
        - *Export Services*
        - *Image Formatting and Transcoding*
    - *Workflow and Data Processing Module*
        - *Response Processors*
            - *Facet Processors*
            - *Field Processors*
            - *Filter Processors*
            - *General Response Processors*
        - *Query Processors*
    - *Social and Collaboration Tools*
- *App Studio Gallery*
    - *Button*
    - *Checkbox*
    - *Color*
    - *Comment*
    - *Field*
    - *Form*
    - *Grid*
    - *Icons*
    - *Input*
    - *Loader*
    - *Message*
    - *Pagination*
    - *Panel*
    - *Progress Bars*
    - *Result List*
    - *Table*
    - *Tabs and Toggle Panes*
    - *Tag*
    - *Timeline*
    - *Tooltip*
    - *Typography*
- *FAQ*
    - *App Studio Versions*
    - *Devices and Browser Support*
    - *Empty Facets in Solr*
    - *Performance*
    - *Responsive Design*
    - *Space for Log Files*
- *Deployment*
    - *Building from Source*
- *Search Platforms*
    - *Solr*
        - *Configuring the Hierarchical Facet Service*
        - *Indexing Hierarchical Taxonomies*
        - *Query Suggestions*
        - *SolrCloud*
    - *Elasticsearch*
    - *Google Custom Search*
    - *Lucidworks Fusion*
    - *Offline*
    - *Pipl*
    - *Shared Platform Attributes*
    - *Twitter*
    - *Wikipedia*
    - *Wolfram|Alpha*
    - *YouTube*
- *Filters*
    - *Filter Syntax*
    - *actual*
    - *actualValue*
    - *dateFormat*

    - *deepFilter*
    - *display*
    - *displayValue*
    - *encodeURIComponent*
    - *facet*
    - *fact*
    - *field*
    - *filterCount*
    - *filterValue*
    - *formatAsCssName*
    - *getUrlForQuery*
    - *hasFilter*
    - *htmlToPlainText*
    - *maxCharacters*
    - *numberFormat*
    - *orderByValue*
    - *placeholder*
    - *property*
    - *pyramidSort*
    - *rangeDateFormat*
    - *result*
    - *roundedNumber*
    - *translate*
    - *trusted*
- *Installation*
    - *Open Source Libraries*
    - *Software Requirements*
    - *Supported Devices and Browsers*
- *Licensing*
- *Styles*
- *Tags*
    - *advanced:button*
    - *advanced:date-range*
    - *advanced:text*
    - *block:interruptor*
    - *chart:credit*
    - *chart:display*
    - *chart:export*
    - *chart:gauge*
    - *chart:histogram*
    - *chart:legend*
    - *chart:tagcloud*
    - *chart:timeline*
    - *chart:tooltip*
    - *class:toggle*
    - *click:outside*
    - *collaborate:bookmark*
    - *collaborate:bookmark-list*
    - *collaborate:comment-form*
    - *collaborate:comment-list*
    - *collaborate:like*
    - *collaborate:saved-query-list*
    - *collaborate:topic-form*
    - *collaborate:topic-list*
    - *content:boolean*
    - *content:form*
    - *content:input*
    - *content:radio*
    - *content:submit*
    - *content:textarea*
    - *controller*
    - *default:src*
    - *dev:track-digests*
    - *drawer*
    - *dynamic:styling*
    - *esri:heatmap*
    - *esri:webmap*
    - *facet:dropdown*

- *facet:hierarchical*
- *geo:heatmap*
- *geo:highmaps*
- *geo:layer*
- *geo:map*
- *graph:flow*
- *helper:constant*
- *helper:set*
- *helper:title*
- *hide:when*
- *instant:result-list*
- *layout:animate*
- *layout:block*
- *layout:box*
- *layout:center-content*
- *layout:content*
- *layout:controls*
- *layout:grid*
- *layout:include*
- *layout:logo*
- *layout:sidebar*
- *lifecycle:delay*
- *lifecycle:display*
- *lifecycle:event*
- *lifecycle:on*
- *lifecycle:required*
- *media:csv*
- *media:excel*
- *media:image*
- *ng:compile*
- *ng:inject*
- *padding:bottom*
- *padding:left*
- *padding:right*
- *padding:top*
- *pivot:table*
- *progress:bar*
- *progress:radial*
- *query:custom*
- *query:filter*
- *query:suggestions*
- *search:advanced*
- *search:alert*
- *search:box*
- *search:breadcrumbs*
- *search:conditional*
- *search:facet*
- *search:facet-list*
- *search:field*
- *search:fields*
- *search:filter*
- *search:filter-list*
- *search:keyword*
- *search:load*
- *search:message-list*
- *search:no-results*
- *search:offcanvas-toggle*
- *search:pagination*
- *search:platform*
- *search:query*
- *search:response*
- *search:response-statistics*
- *search:result*
- *search:result-list*
- *search:results-per-page*
- *search:sort*
- *search:tabs*
- *search:typeahead*

        - *security:user-actions*
        - *series:facet*
        - *series:pivot*
        - *series:result-list*
        - *series:time*
        - *show:when*
        - *slide:toggle*
        - *social:comment-form*
        - *social:comment-list*
        - *styling*
        - *table:body*
        - *table:cell*
        - *table:header*
        - *table:header-cell*
        - *table:result-list*
        - *table:row*
        - *task:form*
        - *task:list*
        - *toggle:controls*
        - *toggle:pane*
        - *toggle:panes*
        - *tooltip*
        - *track:clicks*
        - *track:query*
        - *trigger:event*
        - *url:tabs*
        - *widget:carousel*
        - *widget:date-range*
        - *widget:datetimepicker*
        - *widget:file-hierarchy*
        - *widget:icon*
        - *widget:in-view*
        - *widget:license*
        - *widget:login-form*
        - *widget:login-page*
        - *widget:logout*
        - *widget:modal*
        - *widget:open-modal*
        - *widget:popover*
        - *widget:scroll-to*
        - *widget:slide-down*
        - *widget:slider*
        - *widget:spelling-suggestions*
        - *widget:spinner*
        - *widget:tutorial*
- *App Studio Concepts*
    - *App Development*
    - *App Studio Enterprise*
        - *App Studio Enterprise Downloads*
        - *Get Started with ASE*
    - *Appkit*
        - *Appkit Concepts*
        - *Appkit tags*
        - *Architecture and Technology Stack*
        - *Data Sources*
        - *Routing and URL Rules*
        - *Signals in Appkit*
        - *Social and Collaboration Tools*
        - *User Interface Components*
    - *Configuration*
        - *Basic Configuration*
        - *Configuration Files*
        - *Externalising Configuration*
        - *Localization Keys*
        - *Overlaying Configuration Values*
        - *Structure of a Web Application*
    - *Navigation*
        - *Breadcrumbs tag*

        - *Faceted Navigation*
        - *Tabs tag*
      - *Rendering Search Results*
        - *Results Table*
          - *Cell tag*
          - *Header Cell tag*
          - *Header tag*
          - *Row tag*
          - *Table Body tag*
          - *Table Result List tag*
        - *Result List tag*
        - *Result tags*
      - *Extending Appkit*
        - *Collaboration Events*
        - *Developing Platforms*
        - *Developing Processors*
        - *Developing Services and Servlets*
      - *Layout and Styling*
        - *Layout of the Default View*
        - *Themes and Styling*
        - *Topic Pages*
    - *Appkit SDK Release Notes*
      - *Appkit SDK 4.14.0 Release Notes*
      - *Appkit SDK 4.13.0 Release Notes*
      - *Appkit SDK 4.11.0 Release Notes*
      - *Appkit SDK 4.10.0 Release Notes*
      - *Appkit SDK 4.9.0 Release Notes*
      - *Appkit SDK 4.8.0 Release Notes*
      - *Appkit SDK 4.7.0 Release Notes*
      - *Appkit SDK 4.6.0 Release Notes*
      - *Appkit SDK 4.5.0 Release Notes*
      - *Appkit SDK 4.4.0 Release Notes*
      - *Appkit SDK 4.3.0 Release Notes*
      - *Appkit SDK 4.2.5 Release Notes*
      - *Appkit SDK 4.2.4 Release Notes*
      - *Appkit SDK 4.2.3 Release Notes*
      - *Appkit SDK 4.2.2 Release Notes*
      - *Appkit SDK 4.2.1 Release Notes*
      - *Appkit SDK 4.2.0 Release Notes*
  - *Query Stage SDK*
    - *Java SDK Library*
    - *Query Stage SDK Javadocs*
    - *Example Query Stage Plugin*
  - *Jupyter Support in Fusion*
- *Release Notes*
  - *Fusion 5.12.0*
  - *Fusion 5.11.0*
  - *Fusion 5.10.0*
  - *Fusion 5.9.3*
  - *Fusion 5.9.2*
  - *Fusion 5.9.1*
  - *Fusion 5.9.0*
  - *Fusion 5.8.1*
  - *Fusion 5.8.0*
  - *Fusion 5.7.1*
  - *Fusion 5.7.0*
  - *Fusion 5.6.1*
  - *Fusion 5.6.0*
  - *Fusion 5.5.2*
  - *Fusion 5.5.1-3*
  - *Fusion 5.5.1-2*
  - *Fusion 5.5.1-1*
  - *Fusion 5.5.1*
  - *Fusion 5.4.5*
  - *Fusion 5.5.0*
  - *Fusion 5.4.4*
  - *Fusion 5.4.3*
  - *Fusion 5.4.2*

- *Fusion 5.4.1*
- *Fusion 5.4.0*
- *Fusion 5.3.5*
- *Fusion 5.3.4*
- *Fusion 5.3.3*
- *Fusion 5.3.2*
- *Fusion 5.3.1*
- *Fusion 5.3.0*
- *Fusion 5.2.2*
- *Fusion 5.2.1*
- *Fusion 5.2.0*
- *Fusion 5.1.5*
- *Fusion 5.1.4*
- *Fusion 5.1.3*
- *Fusion 5.1.2*
- *Fusion 5.1.1*
- *Fusion 5.1.0*
- *Fusion 5.0.2*
- *Fusion 5.0.1*
- *Fusion 5.0.0*
- *Deprecations and Removals*

About Legacy Docs

This site includes legacy documentation for product versions that are no longer supported, according to our Lucidworks Fusion Product Lifecycle *(https://doc.lucidworks.com/policies/7shln5/lucidworks-version-support-lifecycle)*.

Legacy documentation is not updated or maintained. As a result, you may discover mistakes or inaccuracies within the documentation.

Recent migrations

Product	End of support	Migrated to legacy
Fusion 5.6.x	February 10, 2023	October 21, 2024
Fusion 5.7.x	June 13, 2023	October 21, 2024
Fusion 5.8.x	September 22, 2023	October 21, 2024
Managed Fusion 5.6.x	February 10, 2023	Managed Fusion documentation is not migrated.
Managed Fusion 5.7.x	June 13, 2023	Managed Fusion documentation is not migrated.
Managed Fusion 5.8.x	September 22, 2023	Managed Fusion documentation is not migrated.

Frequently asked questions

Why did you migrate legacy docs to a new site?

A new site is the best way to provide a high quality user experience. The legacy documentation site is based on this site and will continue to receive feature updates, as appropriate.

What about my saved pages?

We understand that you may have saved pages in bookmarks, shared documents, and other places. To help minimize the impact of this migration, we added page redirects from the current documentation site. However, we recommend that you update your saved pages as soon as possible.

Will more products be affected in the future?

Yes, we plan to migrate documentation for product versions that reach their end of life support to the legacy documentation site in the future. This will take place on or after the date stated in the Lucidworks Fusion Product Lifecycle *(https://doc.lucidworks.com/policies/7shln5/lucidworks-version-support-lifecycle)* policy.

Introduction to Fusion

Fusion combines the Apache Solr open-source search engine with the distributed power of Apache Spark for artificial intelligence. Highly scalable, Fusion indexes and stores data for real-time discovery.

Fusion UI Tour

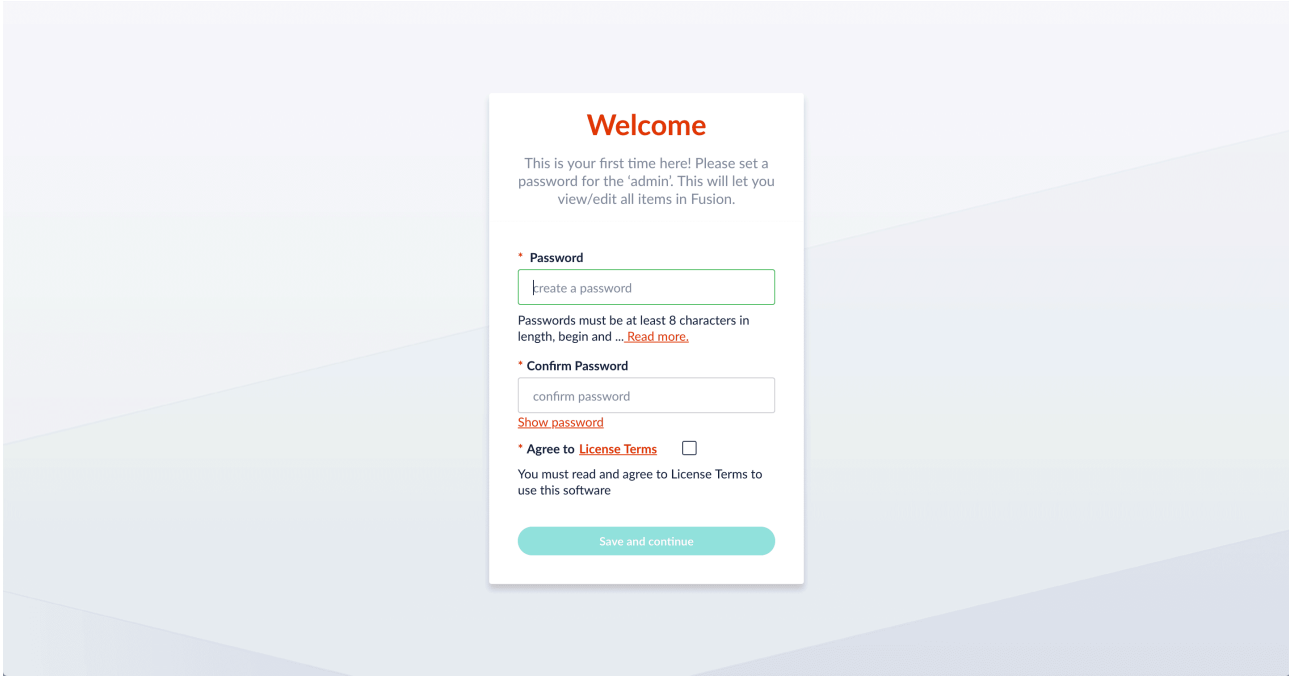
The Fusion UI gives you easy access to all Fusion features.

The UI service utilizes the API service, so each UI feature has an API equivalent. For programmatic access to the features you see in the UI, see the *API Reference Guide*.

Logging in

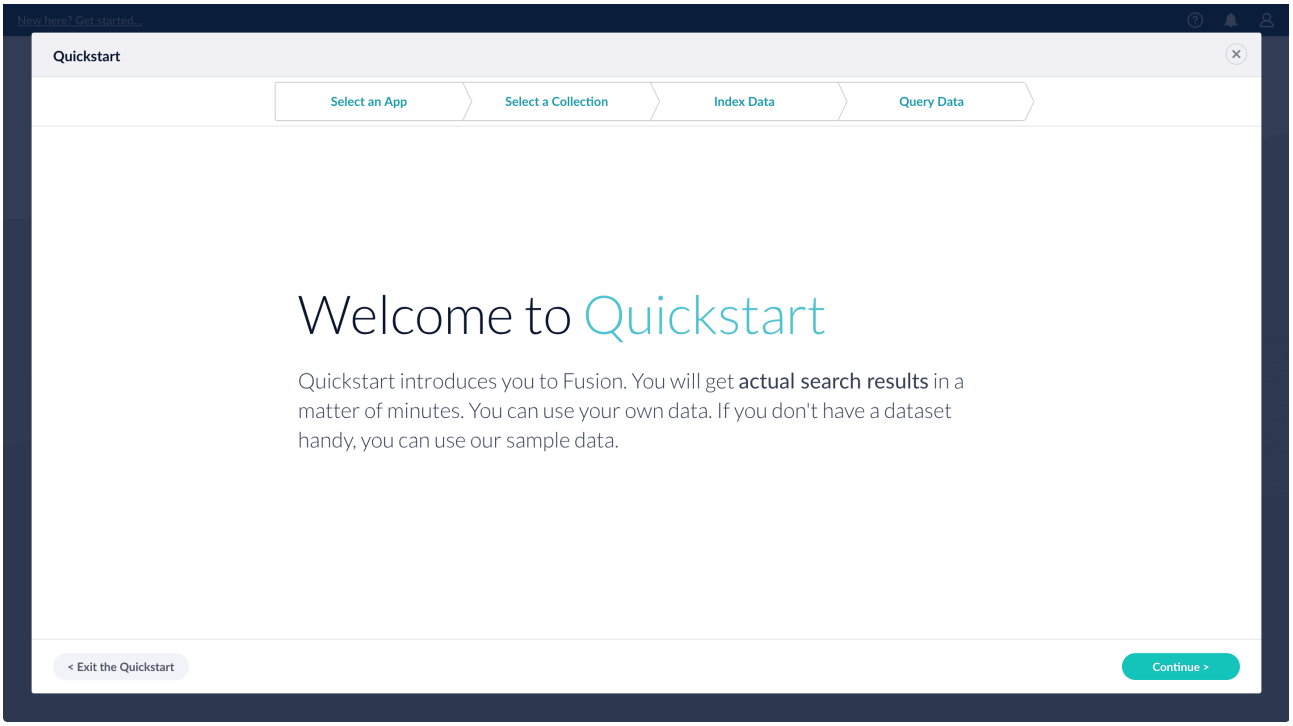
The Proxy service runs on port 6764 by default. Access the Fusion UI through the Proxy service. To log in, point your browser to this port on the Fusion host, for example, `http://FUSION_HOST:FUSION_PORT/`.

When you are the *first user* to log in:
Fusion prompts you to set the password for the "admin" account and agree to the license terms. All other accounts must be created by an admin.



The first time you log in:
Fusion opens the *Quickstart*, to give you a demonstration of Fusion's most basic features. You can click **Exit the Quickstart** to close it and go to the Fusion launcher.

You can open the Quickstart at any time by navigating to the Fusion launcher and clicking **New here? Get started…**.



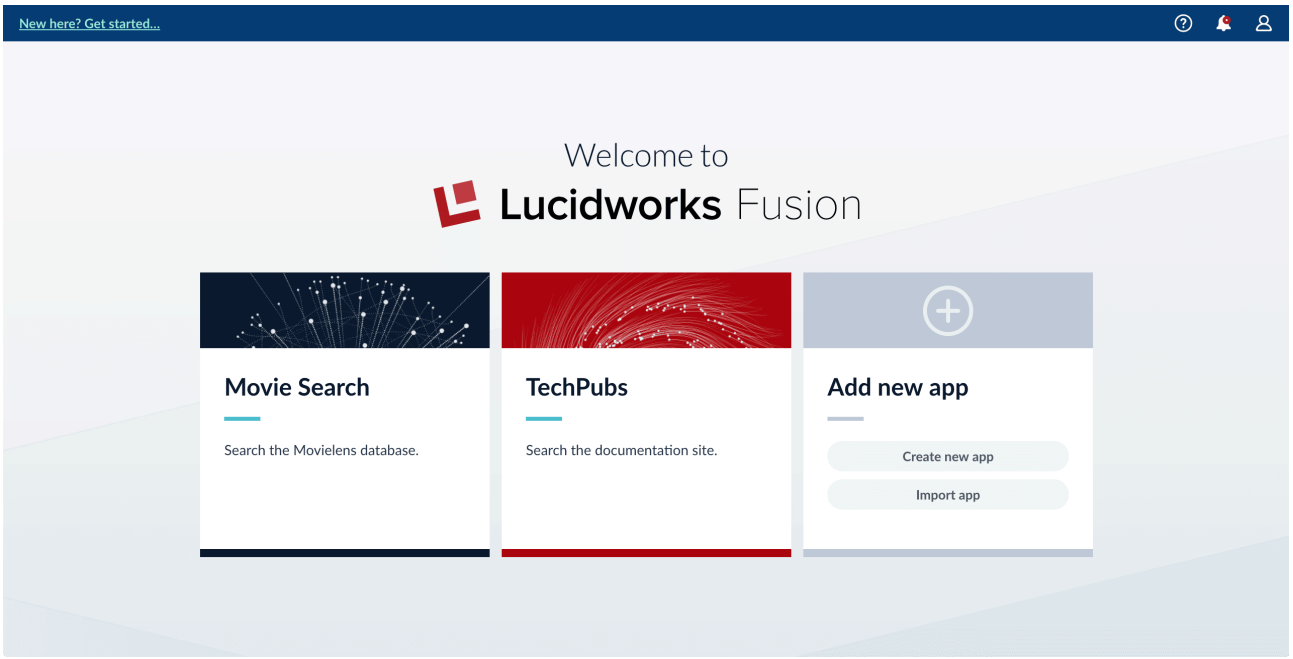
The brief tour below shows you how to find your way around the interface. For a more detailed tour, try the *Get Started* tutorial.

A brief tour

- The *launcher* is where you can view all apps in your Fusion environment and create new ones.
- The *Fusion workspace* is where you configure how an app ingests, indexes, queries, and analyzes data.

The Fusion launcher

When you log in to Fusion, the first page is the Fusion launcher. This is where you create and launch apps:



To return to the Fusion workspace, hover over , and then click **Return to launcher**.

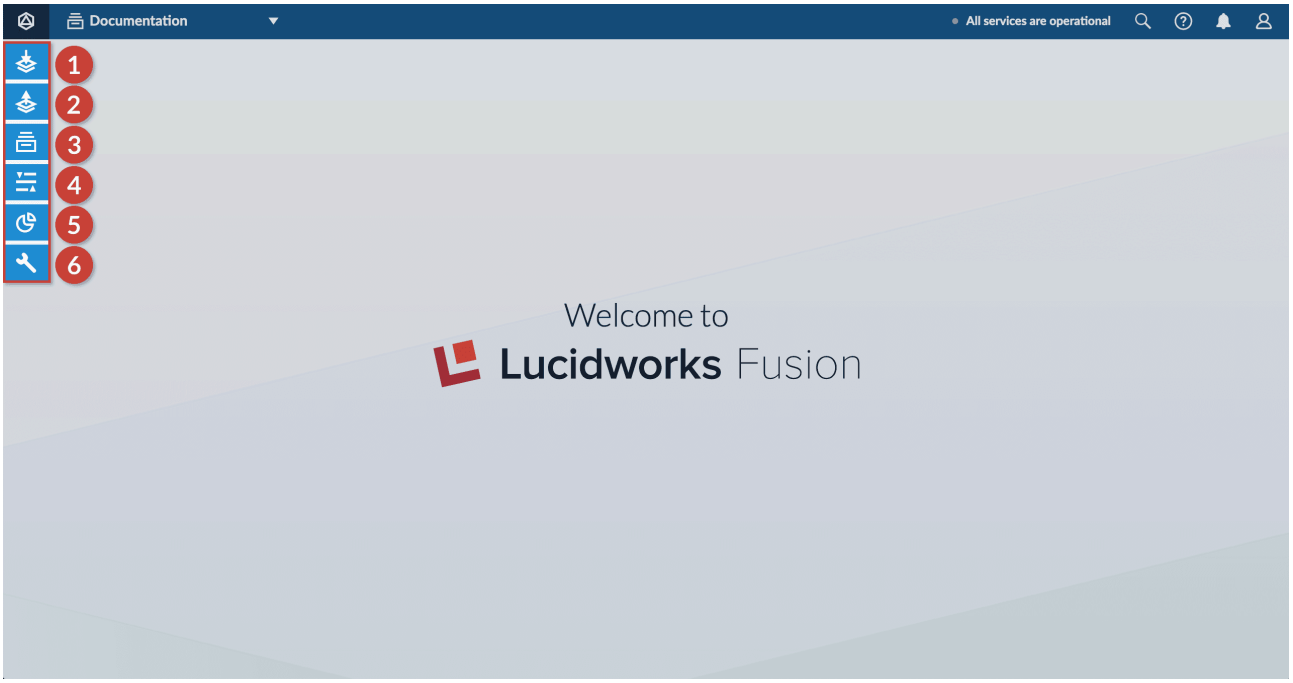
See details about *creating an app*.

The Fusion workspace

You can reach the Fusion workspace by clicking on any app in the launcher. When you enter the Fusion workspace, you find all the tools you need to configure the selected app:

- On the left side are the *app tool menus*, where you can find tools for working with your app.
- *System-wide tools* are at the top of the UI. With the exception of the **Collections** tool, which allows you to switch between collections for your app, these tools do not concern your app specifically.

App tool menus

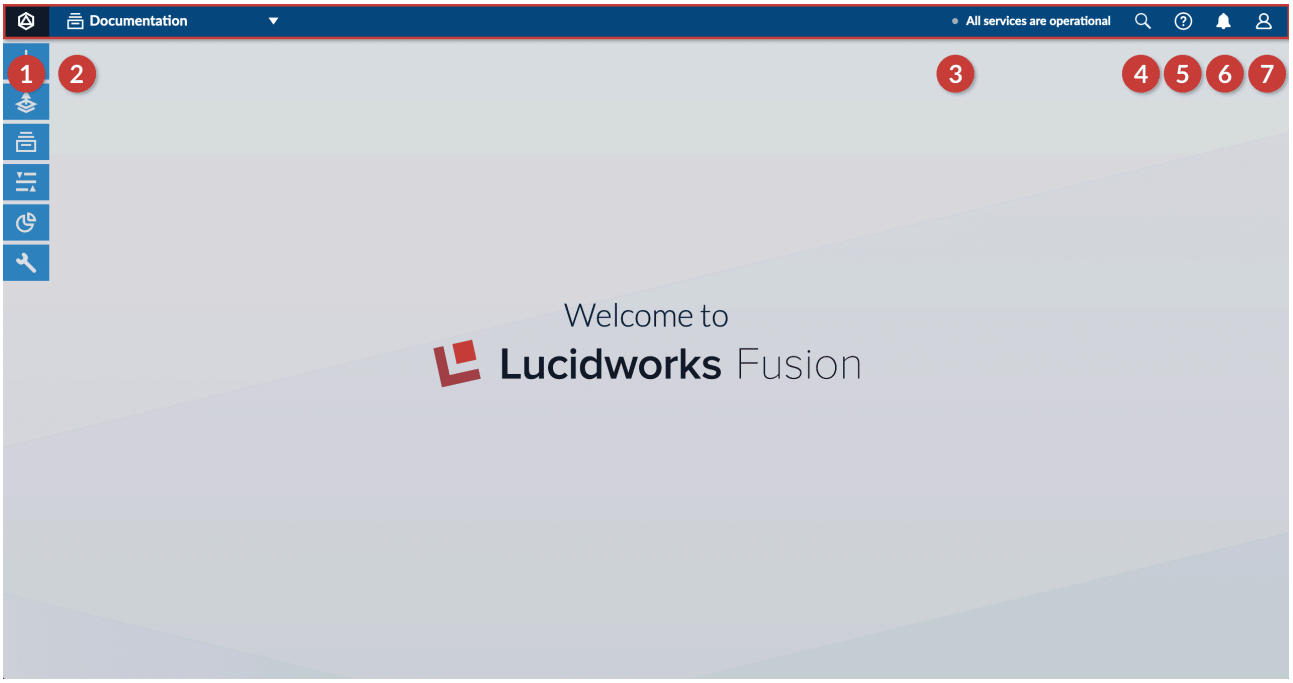


On the left side of the screen are six menus where you can find tools for working with your app:

1. **Indexing** (getting data in)
   The *Index Workbench* is the primary tool for getting data into your app through the Fusion UI. It provides an all-in-one view of the components of a datasource. You can also access those components individually through this menu.
2. **Querying** (getting data out)
   For getting data out, the *Query Workbench* is the main tool. It shows you a live preview of search results while you configure and fine tune your query pipelines.
   See also *Getting Data Out*.
3. **Collections**
   Manage your data collections and configure their features.
   See also *Collection Management*.
4. **Relevance**
   Add or edit rules, recommendations, and signals.
5. **Analytics**
   View App Insights, run Experiments, or view your Dashboard.

6. **System**
   Manage your system, including access control and logs.

System-wide



On the right are menus for system-wide actions:

1. **Apps**
   Switch between apps or return to the app launcher.
2.

Collections
   View and navigate to other collections.
3.

Microservices
   View the status of all Fusion microservices.
4. **Search**
   Click the search icon and enter a query to find areas of the Fusion UI, objects, and related documentation. Clicking an object opens the *Object Explorer*.
5. **Help**
   Access the documentation.
6. **Alerts**
   New alerts are briefly displayed in the foreground. Open this menu to see past alerts, or to clear the alert history.

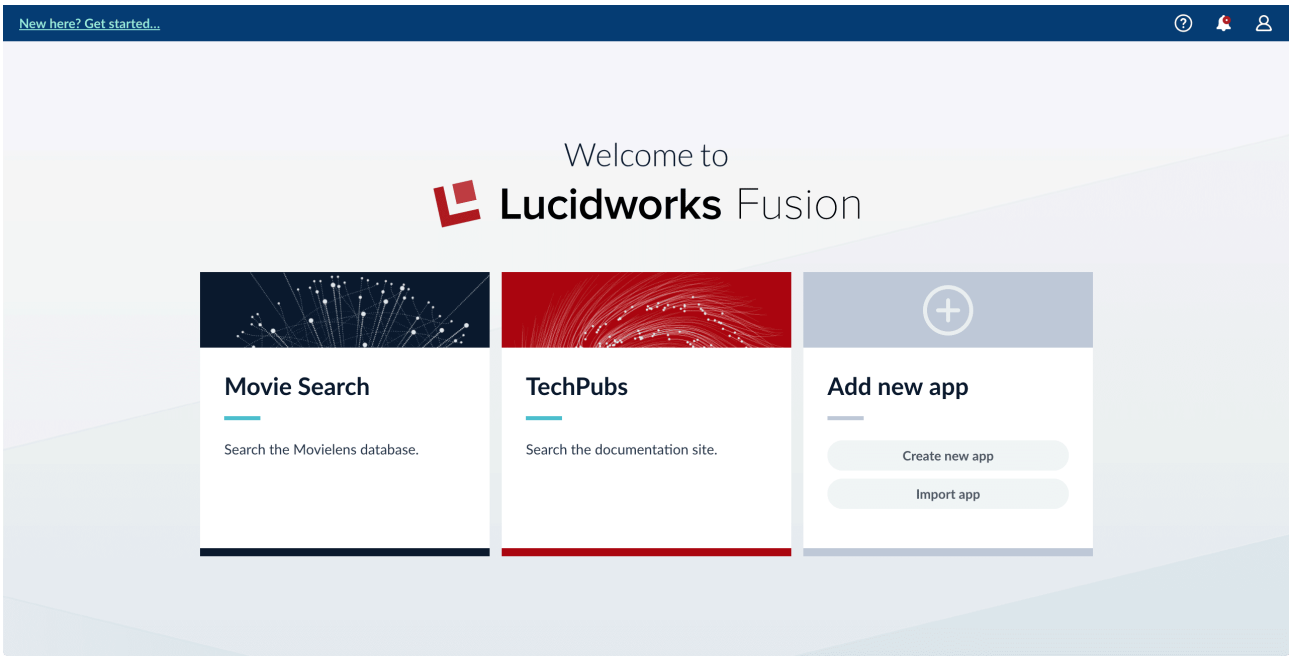
7. **Account Settings**
   Edit your profile, log out, or manage your Fusion license.

Fusion Apps

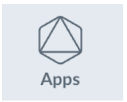
Fusion apps provide tailored search functionality to specific groups of users.

An app is a named set of linked *objects*, including collections, datasources, index and query pipelines, index and query profiles, parsers, and more. You can define security on a per-app basis using *roles* and *security realms*.

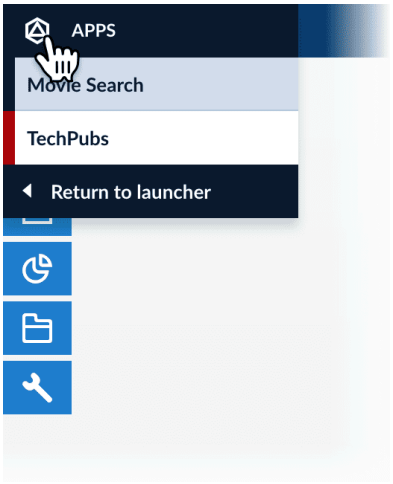
Generally, you will create multiple apps for different purposes. You can view all of your apps in the launcher when you log in to the Fusion UI:



To enter the Fusion workspace for any app, click the app name.

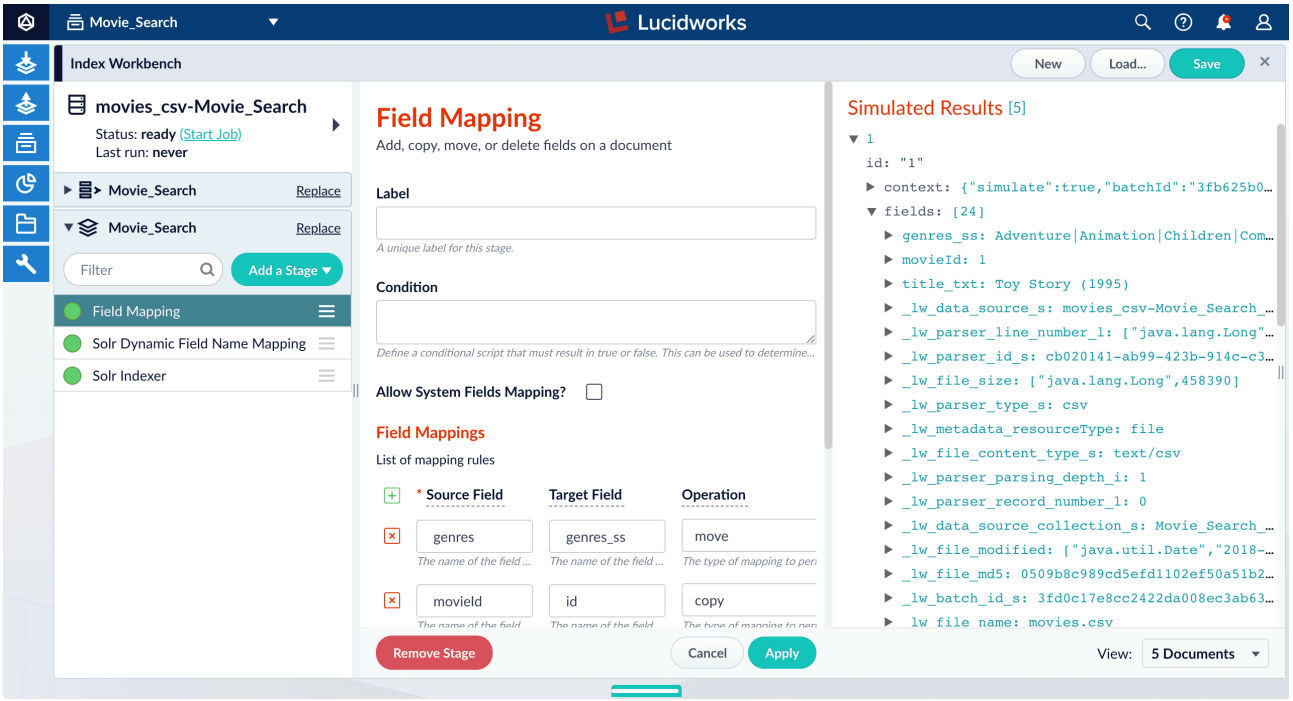
In the Fusion workspace, you can hover over 

to switch to another app, create a new app, or return to the launcher:



Index Workbench

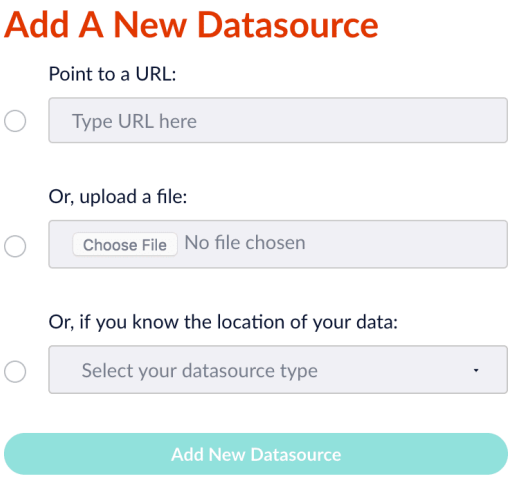
The Index Workbench is a powerful tool that combines key aspects of the data indexing configuration process into one user-friendly panel. It guides the user through the workflow for configuring datasources, parsers and index pipelines, and then running a datasource job to index your data.



Datasources

A datasource is a complete data ingest configuration that consists of the following components:

- **Connector.** Connectors are out-of-the-box components for pulling your data into Fusion. Lucidworks provides a wide variety of connectors, each specialized for a particular data type. For a list of connectors, see *Fusion Connectors*.
- **Parser.** Parsers provide fine-grained configuration for inbound data. See *Parsers* for more information.
- **Index pipeline.** Index pipelines transform incoming data for indexing by Fusion. See *Index Pipelines* for more information.

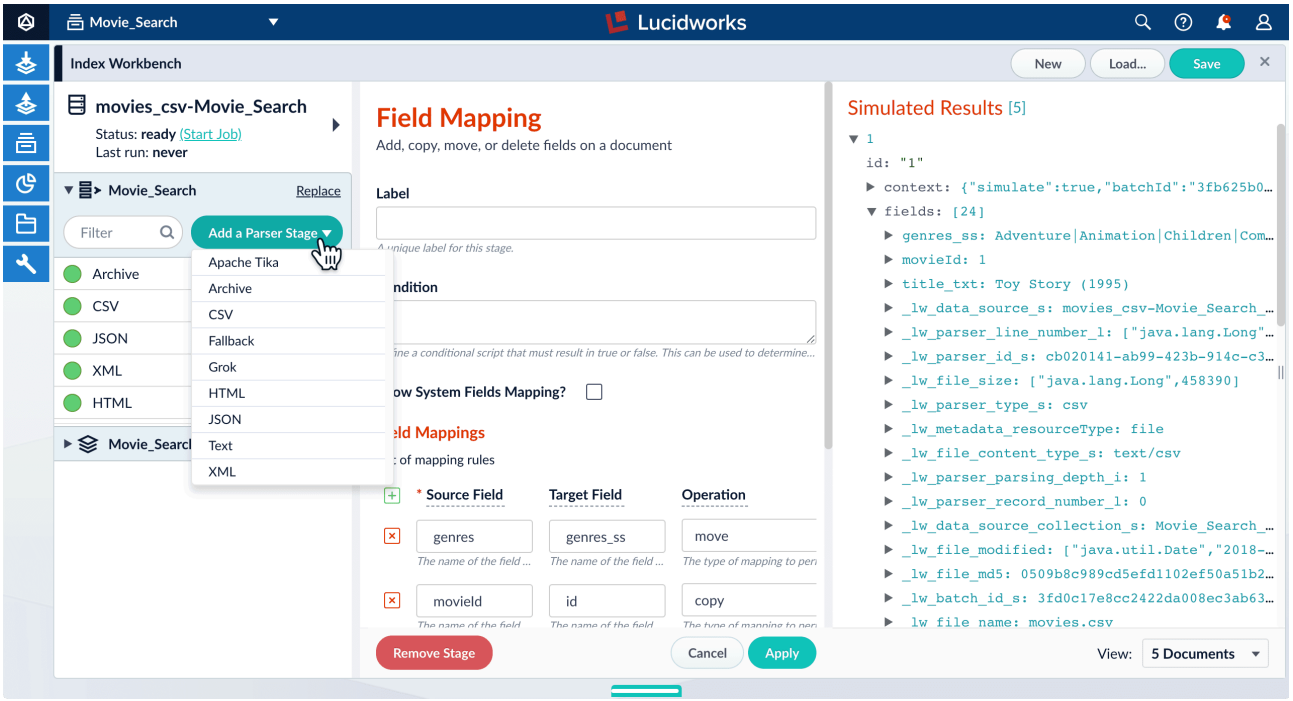


To set up a new datasource in the Index Workbench, click **New**. The **Add A New Datasource** panel appears, offering three simplified methods of configuring how Fusion fetches your data:

- By URL
- By local file upload
- By selecting an installed connector to configure
  See *Connectors Configuration Reference* for a complete list of available connectors. Some are installed by default, and others require separate download and installation.

Parsers

Parsers are a configurable component of the indexing workflow, for flexibility and specificity when parsing inbound data.



A parser consists of an ordered list of parser stages that is completely customizable. The same parser stage can be added to a given configuration multiple times if the different specified settings within those stages best suits the parsing of the data. There is no limit to the number of parser stages that can be included in a parser, and the order in which they run is also completely flexible.

In a parser, after all of the doctype-specific parser stages have run, the Tika and Fallback stages are useful catch-all stages that can attempt to parse anything that has not yet been matched. Tika is used for parsing many types of unstructured documents like PDFs, DOCX, and many more. If all of the other stages in the parser fail to completely parse the data, the Fallback stage can copy the raw bytes directly to Solr.

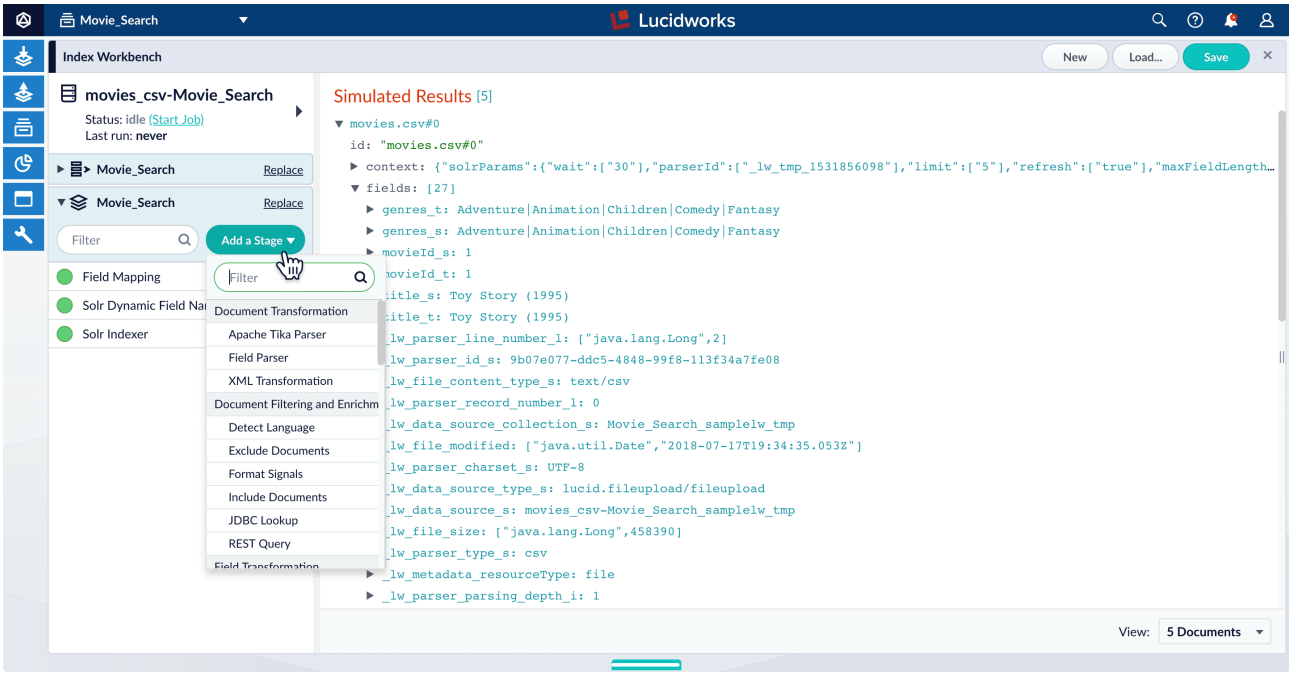
Index pipelines

An *Index Pipeline* transforms incoming data into a document suitable for indexing by Solr via a series of modularized operations called stages. Fusion provides a variety of specialized index stages to index data effectively. Stages can be selected, configured, and enabled or disabled in the Index Pipeline section of the Index Workbench.

Once you finish configuring a datasource using the Index Workbench, you can move on to setting up queries using the *Query Workbench*, which provides a similar workflow for configuring and previewing search results.

Adding a pipeline stage

Click **Add a Stage** to add index pipeline stages that can perform data filtering, transformations, and more.



The *Solr Indexer stage* is the only pipeline stage that is required to complete the indexing process, and therefore exists in every query pipeline. It is always the last stage in a series.

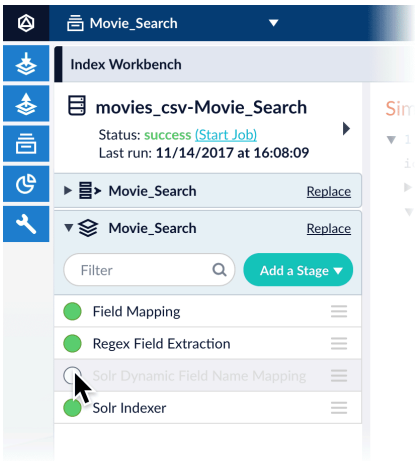
Re-ordering pipeline stages

The order of the pipeline stages matters, because the output from one stage becomes the input to the next stage. For example, the *Solr Indexer stage* must always come last in the sequence, so that data is indexed only after it has been processed by all other stages. Putting this stage first in the sequence means that subsequent stages have no effect on the indexed data.

Click and drag any stage in the pipeline to move it up or down in the sequence of stages. The preview panel automatically updates the results to reflect the output of the new sequence.

Enabling and disabling pipeline stages

By default, every stage in an index pipeline is enabled. While working with a pipeline, it can be helpful to disable stages without removing them completely. This allows you to preserve a stage's configuration while observing how the search results change in its absence. You can re-enable the stage at any time. When you save a query pipeline, the enabled/disabled state of each stage is also saved.

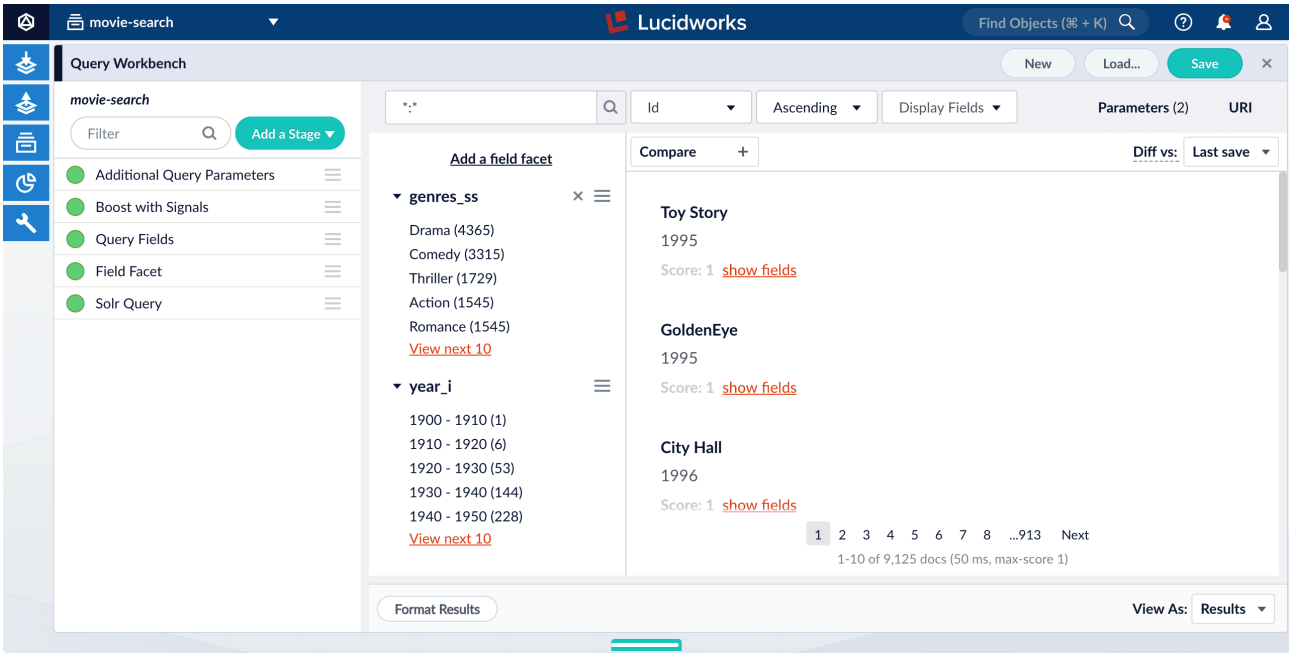


Click the circle next to any stage in order to enable or disable it.

Query Workbench

The Query Workbench is where you edit query pipeline stages, results parameters, and more, then preview the impacts of the changes in real time before saving them. Relevancy tools such as boosting and blocking can be accessed and altered through the Query Workbench. The Query Workbench streamlines the process of querying indexed results and fine-tuning Query Pipelines to surface the results that best satisfy the end user's needs.

See the *Query Language Cheat Sheet* for help constructing queries.

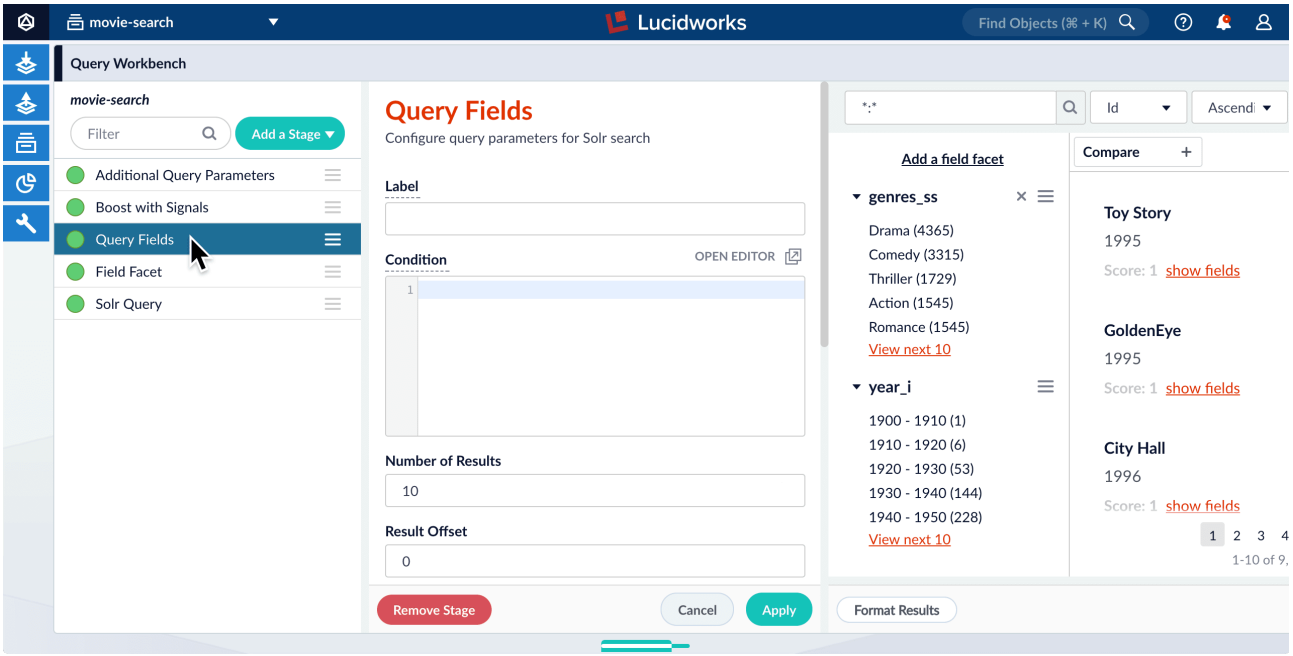


With this sandbox paradigm, you can experiment with query pipelines without consequence because Fusion uses a copy of a pipeline to simulate the new results, and it is not permanently modified in the collection until you click **Save**.

Query pipelines and pipeline stages

Query pipelines work by processing search requests and returning an ordered list of matching documents. Each pipeline consists of a series of query stages that can be added, ordered, and configured using the Query Workbench.

Select any stage in the pipeline to open its configuration panel:



Boosting and blocking

As you search your data and inspect the results, you can manipulate the rankings of individual documents. Boosting a document raises its ranking, while blocking a document removes it from search results.

Boosting and blocking affects the results for specific queries. For example, when you search for "citizen" and boost the document for "Citizen Kane", your end users will see that title boosted to the top when they also search for "citizen", but not necessarily when they search for "kane".

The Query Workbench provides convenient buttons for instantly blocking or boosting the documents that you see in the preview panel. Clicking **Boost** automatically adds the *Boost Documents stage* to your query pipeline, with a boost rule that matches your query and the document you clicked. Likewise, clicking **Block** adds and configures the *Block Documents stage*.

Compare mode

Compare mode allows you to compare search results using two different query pipelines, or compare search results from the original query pipeline and the copy being modified in the Query Workbench. On the right is the working pipeline, which you can edit. On the left is one of your saved pipelines.

Object Explorer

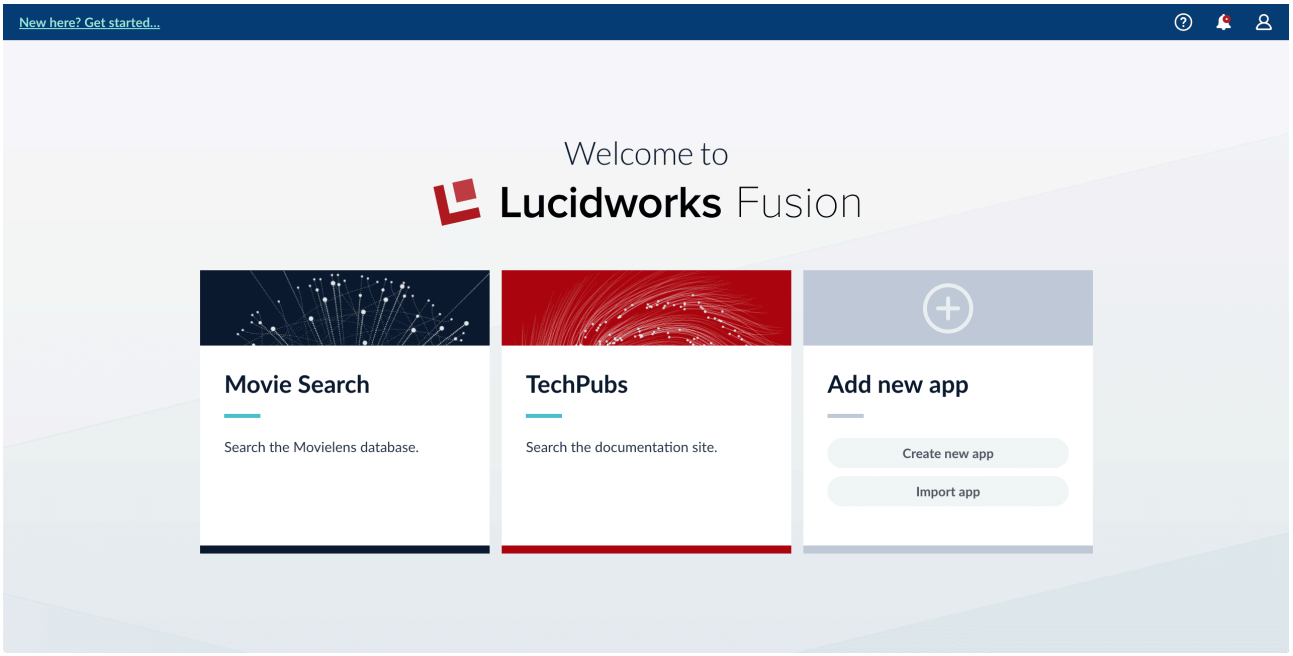
Object Explorer lets you easily find and manage Fusion *objects* within a specific app and across multiple apps in an entire Fusion deployment.

Fusion apps and objects

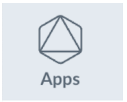
Fusion apps provide tailored search functionality to specific groups of users.

An app is a named set of linked *objects*, including collections, datasources, index and query pipelines, index and query profiles, parsers, and more. You can define security on a per-app basis using *roles* and *security realms*.

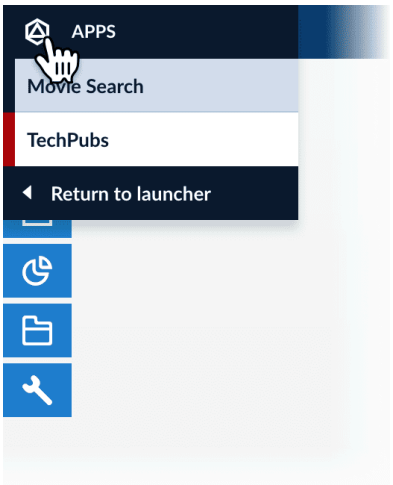
Generally, you will create multiple apps for different purposes. You can view all of your apps in the launcher when you log in to the Fusion UI:



To enter the Fusion workspace for any app, click the app name.

In the Fusion workspace, you can hover over 

to switch to another app, create a new app, or return to the launcher:



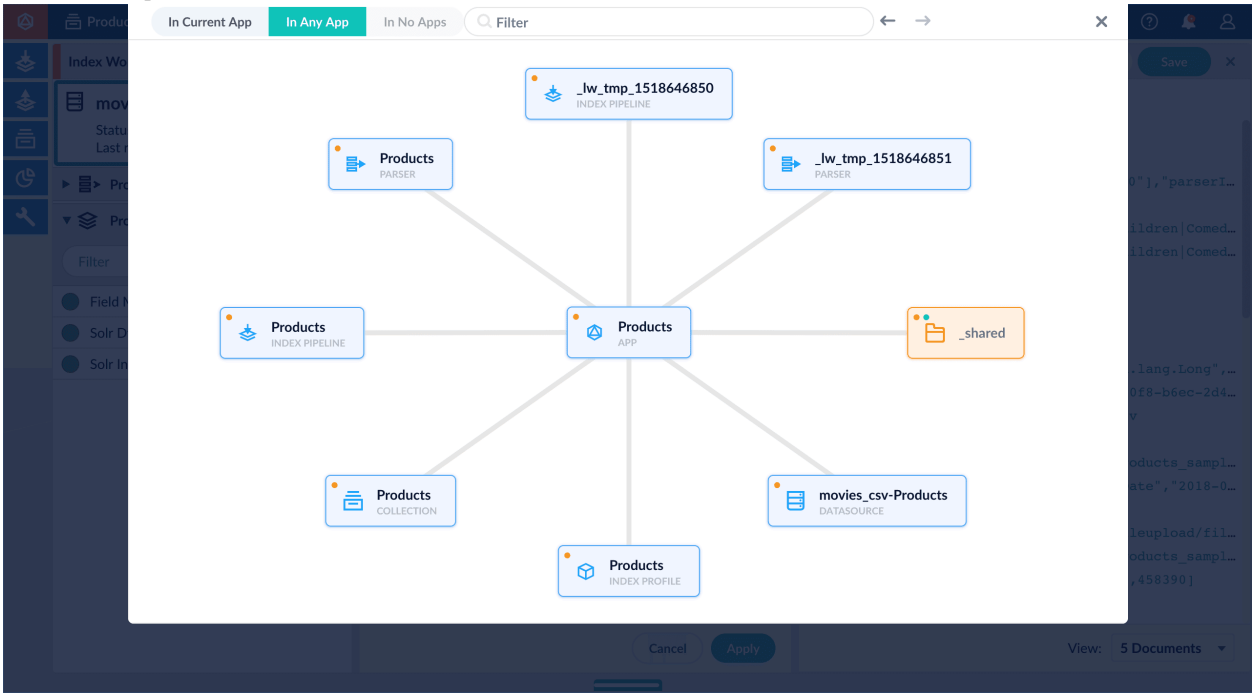
An object is something that has a specific function in a Fusion app, such as an index pipeline or Spark job. It is often something that you configure. An app is an object. Several object types are *groups* that group other objects. For more information about object types, see *Object types*.

What you can do in Object Explorer

In Object Explorer, you can explore objects.

Which objects you can explore depends on the *permissions* that your security realm and user definition grant you. As the user `admin`, you can explore all objects supported by Object Explorer.

- **Determine which objects are in which apps.** You can determine which objects are in the current app, in any app, in no apps, and in any app or no apps. When you navigate to an object, you can display the apps that the object is in.
- **View and navigate between related objects.** In Object Explorer, related objects are connected by lines. Clicking an object moves it to the center of the Object Explorer window and displays related objects around it.
- **View related objects.** When you select an object, Object Explorer centers and displays related objects around it, connected by lines, for example:



- **Perform some management tasks.** You can perform some management tasks inside Object Explorer. For example, you can schedule jobs and tasks from Object Explorer. For other management tasks, Object Explorer takes you to where you need to be.
- **Add and remove objects** (*Advanced feature*) - When you create an app, it consists of objects. When you create additional apps, Fusion can *share* some objects across apps. Shared objects are in the `_shared` system group, and they have multiple dots on them, one for each app they are in.
  In some cases, you might want to remove an object from an app (undo sharing) or add an object to an app. For example, after configuring a parser in one app, you could share the parser with other apps, *without having to share things you do not want to share*, for example, the collection.

Object types

You can explore these types of objects with Object Explorer:

App	Spark job or task	Collection
Datasource	Index pipeline	Parser
Query pipeline	A generic icon that is currently used for query profiles, index profiles, blobs, and experiments	Recommendations group
System group	User-defined group or Fusion-defined group that is not a system group	

The Fusion Stack

In addition to Lucidworks' proprietary technology, the Fusion platform incorporates a powerful stack of open-source software.

- *Apache Pulsar* is a message bus that combines queuing with publishing-subscribing features. A producer publishes a message to a topic and a consumer subscribes to that topic using a subscription type to receive the topic messages.
- *Apache Solr* is the search engine at the core of Fusion. Solr stores your data in a primary collection for the data your users will search and auxiliary collections for signals, aggregations, and related data.
- *Apache Spark* is the cluster-computing framework for large-scale processing jobs. These jobs can be turned into stepwise tasks and distributed across a cluster of computers.
- *Milvus* is a vector-similarity search engine integrated into Fusion. It stores the vectors produced by machine learning models and reduces Solr's workload.
- *ZooKeeper* is a distributed configuration service, synchronization service, and naming registry. Fusion uses a ZooKeeper *microservice* to configure and manage all Fusion components in a single Fusion deployment.

Apache Solr

Solr is the open-source search engine at the core of Fusion. When you *index your data*, it is stored in a Solr collection.

Collections

Your data is organized into collections. When you create an app, Fusion automatically creates a collection with the same name. You can create additional collections in any app.

A primary collection contains the data that your users will search. Every primary collection is associated with a set of auxiliary collections that contain related data, such as signals, aggregations, and more.

Under the hood, a Fusion collection is a distributed index in Solr, defined by a named configuration stored in ZooKeeper, with these properties:

- Number of shards
  Documents are distributed across this number of partitions.
- Document routing strategy
  How documents are assigned to shards.
- Replication factor
  How many copies of each document in the collection.
- Replica placement strategy
  Where to place replicas in the cluster.

If your data is already stored in a Solr instance or cluster, you can manage this collection in Fusion by creating a Fusion collection that imports the existing Solr collection.

	Collection names are case-insensitive, but Fusion preserves case when displaying collection names.

Auxiliary Collections

Every primary collection is associated with a set of auxiliary collections that contain related data, such as signals, aggregations, and more.

Some auxiliary collections are created for every primary collection. Others are created only for the app's default collection, one per app.

Auxiliary collections are described below:

`APP_NAME_job_reports`	Output from Fusion *experiments*, *Ranking Metrics jobs*, and *Head/Tail Analysis jobs*.	1 per app
`APP_NAME_query_rewrite`	A collection of documents to use for *rewriting queries*, optimized for high-volume traffic. These documents originate from the `COLLECTION_NAME_query_rewrite_staging` collection. Certain Fusion query pipeline stages read from this collection: • *Text Tagger* • *Apply Rules* • *Modify Response with Rules*	1 per app
`APP_NAME_query_rewrite_staging`	A collection of documents created by the Rules Editor or by certain *Fusion jobs*, not optimized for production traffic. Documents move from this collection to the `COLLECTION_NAME_query_rewrite` collection as follows: • Job output documents with high confidence contain a `review=auto` field and are moved to the `COLLECTION_NAME_query_rewrite` collection automatically. • Job output documents with low confidence contain a `review=pending` field. When these are approved by a Fusion user, Fusion copies them to the `COLLECTION_NAME_query_rewrite` collection.	1 per app
`COLLECTION_NAME_signals`	A search query logs and signals collection.	1 per collection
`COLLECTION_NAME_signals_aggr`	A collection for aggregated signals.	1 per collection
`APP_NAME_user_prefs`	A collection of data to support App Studio's social features, such as user-generated tags, bookmarks, comments, ratings, and so on.	1 per app

Do not create primary collections with names that end in the suffixes above; these are reserved for Fusion auxiliary collections, which are created and managed by Fusion directly.

Fusion maintains a set of Solr collections that store Fusion's own log files and other internal information. These are called *System Collections*, described below.

Do not create primary collections named "logs" or beginning with "system_". These names are reserved for Fusion system collections.

Fusion uses ZooKeeper to register information about all collections, and the Fusion components and services related to a collection. The Fusion components associated with a collection include:

- Datasources
- Pipelines
- Profiles
- Signals and aggregations
- Analytics dashboards

System Collections

Fusion automatically creates some collections that are used for internal purposes and shared across all apps:

- **system_autocomplete** store the content that the Fusion UI displays when you use the search bar.
- **system_blobs** stores *blobs* in Solr. This is used to store model files for the NLP components and other binary files used by Fusion components.
- **system_history** keeps a record of configuration changes, start and stop times for services and experiments, and more.
- **system_jobs_history** keeps a record of Fusion *jobs*, including start/stop times and status.
- **system_logs** stores parsed Java logs from the REST API, connectors-classic component, and other parts of Fusion, like proxy, connectors-rpc, and Appkit App Insights. It also includes HTTP logs and optional gc logs (off by default).
- **system_messages** is used by Fusion's *messaging services*.

Collection Configuration Properties

Collections have three properties that you can configure only when you are creating a collection using the *Collections API*.

Property	Description	Default behavior
signals *	The `signals` property determines whether to create auxiliary collections with suffixes `_signals` and `_signals_aggr`.	When you create a collection in the Fusion UI, `signals` defaults to **true**. When you create a collection using the Fusion API, this property defaults to **false**.
searchLogs	The `searchLogs` property determines whether to create an auxiliary search query logs collection with suffix `_logs`.	When you create a collection in the Fusion UI, this property defaults to **true**. When you create a collection using the Fusion API, this property defaults to **false**.

*Signals are events with timestamps that can be used to improve search results. For more information about signals in Fusion, see *Signals* in the Fusion documentation.

**In schemaless mode, if a document contains a field not currently in the Solr schema, Solr processes the field value to determine what the field type should be defined as, and then adds a new field to the schema with the field name and field type. This behavior can be convenient during preliminary application development, but it is rarely appropriate in a production environment.

Using profiles to associate collections with pipelines

Index pipelines and query pipelines are not connected to a specific collection by default. Index profiles and query profiles are configurations that create consistent endpoints for indexing and querying, each with a specific pipeline and collection.

- *Index Profiles* work with index pipelines for getting content into the system.
- *Query Profiles* work with query pipelines for user queries.

Field Editor UI

The Fusion UI includes a space under Collections to edit Fields. The Fusion UI includes a space under Collections to edit Fields. Descriptions for these fields can be found in the Field Type Definitions section of the *Solr Reference Guide* associated with your Fusion release.

Field options displayed in the UI include:

- **Dynamic** checkbox (cannot change via UI)
- **Field Name** (cannot change via UI)
- **Field Type** (a preset value is shown that can be changed using edit mode)
- Checkboxes for **Indexed**, **Stored**, **Multivalued**, **Required**
- Text field to enter a **Default Value**
- **Copy Fields** uses the plus sign to add rows (**static** can copy to `raw_content` or `text`; **dynamic** can copy to any `raw_content`/`text` or any other dynamic field)
- **Advanced** toggles checkboxes for **Doc Values**, **Omit Norms**, **Omit Positions**, **Omit term freq and positions**, **Term Vectors**, **Term Positions**, **Term Offsets**

Milvus

Milvus *(https://milvus.io/)* is an open-source vector similarity search engine, integrated into Fusion to streamline its deep learning capabilities and reduce the workload on Solr. See Milvus v0.10.2 documentation *(https://www.milvus.io/docs/v0.10.2/overview.md)*.

Milvus collections

Milvus stores the vectors produced by machine learning models in one or more Milvus collections.

For each collection, the following parameters need to be set:

- `Collection Name`. A name for the Milvus collection you are creating. Milvus requires that collection names only include alphanumeric characters and the underscore `_`.
- `Dimension`. The dimension size of the vectors to store in this Milvus collection. The Dimension should match the size of the vectors returned by the encoding model. For example, the `Smart Answers Pre-trained Coldstart` models output vectors of 512 dimension size. Dimensionality of encoders trained by `Smart Answers Supervised Training` job depends on the provided parameters and printed in the training job logs. See *Smart Answers Supervised model training* for more details.
- `Index file size`. Vectors are added to the raw data files and then indexed. Files larger than this will trigger index building for raw data files.
- `Metric`. The type of metric used to determine how close the search vector is to the vectors in the collection. It is used to calculate the vector similarity scores. In most cases we suggest to use Cosine Similarity which can be obtained by using the `Inner Product` metric over normalized vectors. It produces values between -1 and 1, where a higher value means higher similarity.

The Metric options for Milvus collections are: Euclidean, Inner Product, Hamming, Jaccard, Tanimoto, Substructure, and Superstructure. The following metrics all require that the Dimension is a multiple of 8: Hamming, Jaccard, Tanimoto, Substructure, and Superstructure. Otherwise the Create Milvus Collection job will fail. See Milvus metrics documentation *(https://www.milvus.io/docs/v0.10.2/metric.md#floating)* for more details.

Milvus can be configured to pre-load one or more collections via the Milvus parameter `preload_collection`. Fusion does not currently ship with a method to manage this parameter, but the Milvus API can be used directly (after first setting up port forwarding via `kubectl port-forward milvus-writeable-<hash>-<random> 19121:19121`).

For an example of how to create and utilize a Milvus collection, see *Configure The Smart Answers Pipelines*.

Milvus indexes

When a Milvus collection is large, creating an index can speed up the similarities search. If you do not create an index on a collection then a FLAT index is used. See Milvus index documentation *(https://www.milvus.io/docs/v0.10.2/index.md)* for more details.

When you create an index on a collection, in addition to the `collection name` you will need to specify:

- `Index Type`. The index type must be one of [`ANNOY`, `FLAT`, `HNSW`, `IVF_PQ`, `IVF_SQ8`, `IVF_SQ8H`, `IVFLAT`, `RNSG`]. We recommend you use the Fusion job default value of `HNSW`.
- `Index Parameters`. Parameters used to create an index in Milvus. These required parameters are specific to the chosen `Index Type`. For example, `nlist=4096` for IVF indexes or `M=36` and `efConstruction=500` for HNSW indexes.

	If you create an index for the collection different from FLAT, index specific query parameters have to be added as Search Parameters in the Milvus Query stage. Refer to the Milvus supported indexes *(https://www.milvus.io/docs/v0.10.2/index.md#Indexes-that-Milvus-supports)* for query parameters for each index type.

Milvus management jobs

Fusion ships with several jobs for managing collections and indexes in Milvus:

- *Create Collections in Milvus*
- *Recreate a Milvus Collection*
- *Delete Collections in Milvus*
- *Create Indexes in Milvus*
- *Delete Indexes in Milvus*

Pipeline stages

For more information about using these stages see *Configure the Smart Answers Pipeline (5.3 and later)*.

- The *Encode into Milvus* index pipeline stage invokes a machine learning model to encode a field and store it into Milvus.
- Query pipeline stages:

- *Milvus Query* performs vectors similarity search in Milvus.
- *Milvus Ensemble Query* performs ensemble of different search results to retrieve metadata from Solr.
- *Milvus Response Update* updates response documents with vectors similarity and ensemble scores.

Troubleshooting

Update a Milvus collection

Update and delete semantics are not supported on individual items in a Milvus collection. We recommend that you use a blue/green set of collections: reindex to apply updates to an empty Milvus collection, then swap between collections to apply the updates.

Spark Operations

Apache Spark *(http://spark.apache.org/)* is an open-source cluster-computing framework that serves as a fast and general execution engine for large-scale data processing jobs that can be decomposed into stepwise tasks, which are distributed across a cluster of networked computers.

Spark improves on previous MapReduce implementations by using resilient distributed datasets (RDDs), a distributed memory abstraction that lets programmers perform in-memory computations on large clusters in a fault-tolerant manner.

These topics explain Spark administration concepts in Fusion 5:

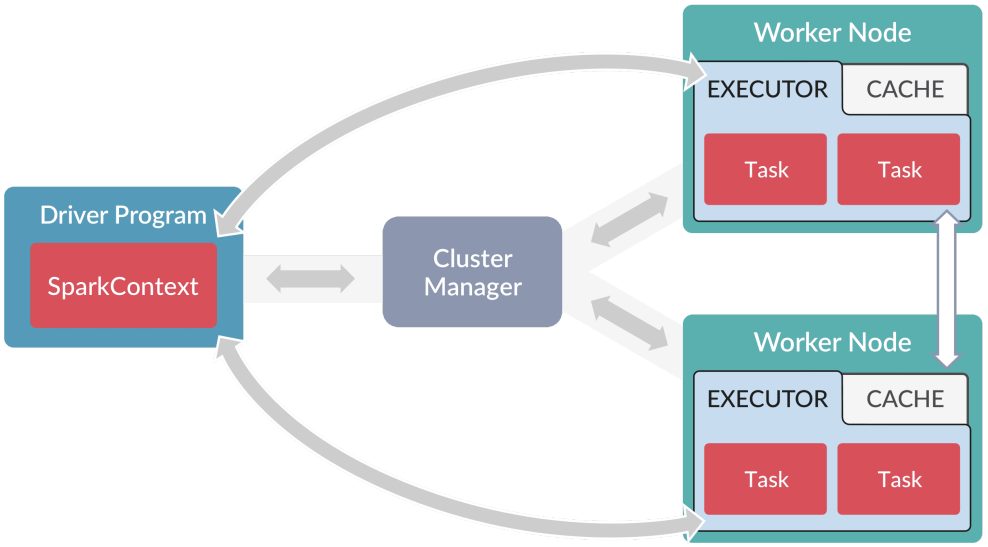
- *Spark Job Drivers*
- *Spark Administration in Kubernetes*

Spark operations include:

1. Audit all *Spark jobs* for natural key support.
2. Audit *SQL Aggregation* jobs for natural key usage.
3. When looking at SQL for the *BPR Recommender job*, audit generated aggregation SQL to ensure that it's using a natural key projection.
4. Support partitioning in all Spark jobs in accordance with config options.
5. Support external data sources for all jobs (Spark, NLP, Clustering, Recommender), including external Spark source support for NLP.

Spark Job Drivers

A Spark "driver" is an application that creates a SparkContext for executing one or more jobs in the Spark cluster. The following diagram depicts the driver's role in a Spark cluster:



In the diagram above, the `spark-master` service in Fusion is the Cluster Manager.

If your Spark job performs any collect operations, then the result of the collect (or collectAsMap) is sent back to the driver from all the executors. Consequently, if the result of the collect is too big too fit into memory, you will encounter OOM issues (or other memory related problems) when running your job.

All Fusion jobs run on Spark using a driver process started by the API service.

Custom jobs

Fusion supports custom Spark jobs that are written in Scala, Java, or Python jobs and that are built using the Spark Job Workbench *(https://github.com/Lucidworks/fusion-spark-job-workbench)*, a toolkit provided by Lucidworks. See the examples in the repository for details.

Drivers

Fusion has four types of job drivers:

- **Default driver.** Executes built-in Fusion jobs, such as a signal aggregation job or a metrics rollup job.
- **Script-job driver.** Executes custom script jobs; a separate driver is needed to isolate the classpath for custom Scala scripts.
- **Spark-shell driver.** Wrapper script provided with Fusion to launch the Spark Scala REPL shell with the correct master URL (pulled from Fusion's API) and a shaded Fusion JAR added. Launched using `{fusion-parent-unix}/fusion/5.4{sub-version}/bin/spark-shell` (on Unix) or `{fusion-parent-windows}\var\fusion\5.4{sub-version}\bin\spark-shell.cmd` (on Windows).
- **Custom-job driver.** Executes custom jobs built using the Spark Job Workbench *(https://github.com/Lucidworks/fusion-spark-job-workbench)*, a toolkit provided by Lucidworks to help build custom Spark jobs using Scala, Java, or Python.

ZooKeeper

Apache ZooKeeper *(http://zookeeper.apache.org/)* is a distributed configuration service, synchronization service, and naming registry. Fusion uses a ZooKeeper *microservice* to configure and manage all Fusion components in a single Fusion deployment.

ZooKeeper Terms

At a high level, ZooKeeper services consist of the following components:

- **znode.** ZooKeeper data is organized into a hierarchal name space of data nodes called znodes. A znode can have data associated with it as well as child znodes. The data in a znode is stored in a binary format, but it is possible to import, export, and view this information as JSON data. Paths to znodes are always expressed as canonical, absolute, slash-separated paths; there are no relative reference.
- **ephemeral nodes.** An ephemeral node is a znode which exists only for the duration of an active session. When the session ends the znode is deleted. An ephemeral znode cannot have children.
- **server.** A ZooKeeper service consists of one or more machines; each machine is a server which runs in its own JVM and listens on its own set of ports. For testing, you can run several ZooKeeper servers at once on a single workstation by configuring the ports for each server.
- **quorum.** A quorum is a set of ZooKeeper servers. It must be an odd number. For most deployments, only 3 servers are required.
- **client.** A client is any host or process which uses a ZooKeeper service.

The Fusion UI and API handle changes to ZooKeeper's configuration. See *the ZooKeeper Import/Export API*.

See the official ZooKeeper documentation *(http://zookeeper.apache.org/doc/current/)* for details about using and managing a ZooKeeper service.

Kubernetes Deployment Architecture

The figure below depicts a Fusion 5 cluster running in a single Kubernetes namespace for high availability:

> The image above is for illustrative purposes only. It depicts services available in Fusion 5.11, but your actual implementation may differ.

Architecture and requirements

- *Which Kubernetes?*
- *Multiple Zones for High Availability*
- *Fusion Microservices*
- *Ingress and Security*
- *Stateless Sessions with JWT*
- *Workload Isolation with Multiple Node Pools*
- *High-Performance Query Processing with Auto-Scaling*

Deploy Fusion on Kubernetes

- *Deploy Fusion 5 in Google Kubernetes Engine*
- *Deploy Fusion 5 in Amazon Elastic Kubernetes Service*
- *Deploy Fusion 5 in Azure Kubernetes Service*
- *Deploy Fusion 5 on Other Kubernetes Platforms*
- *Deploy Fusion at Scale*

Upgrade Fusion on Kubernetes

Upgrade information about Kubernetes is included in the *Fusion 5 Upgrades* topic.

Indexing Data Flow

With Fusion, your data is indexed in a set of collections in Fusion's Solr core.

A *primary collection* holds your searchable content, such as your product catalog, knowledge base, blog articles, product reviews, and so on.

A set of *secondary collections* are associated with your primary collection to hold related data that Fusion can use to enhance the relevancy of your search results.

This topic shows you how different types of data flow through Fusion to be indexed in your primary and secondary collections.

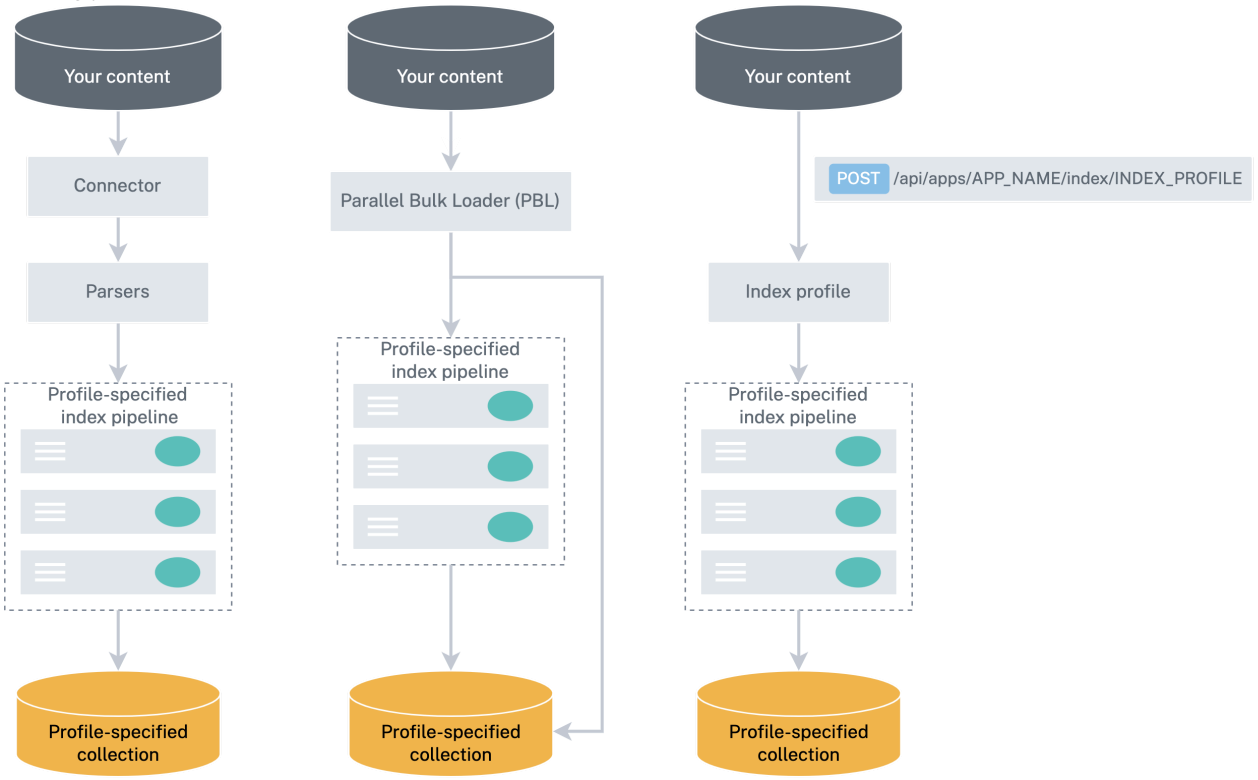
1. Index your content

No matter where your content is located, Fusion can index it. The Fusion collection where your searchable content is indexed is called the primary collection. Secondary collections are automatically created to hold related data, such as signals, machine learning job output, and so on.

There are a few ways to index your searchable content to the primary collection:

- *Connectors*
  Lucidworks has a wide variety of connectors for many types of data sources. *Find your connector*.
  Once the connector fetches your data, *parsers* read it before passing it to the *index pipeline*.
- The *Parallel Bulk Loader (PBL)*
  The PBL can send your data to an index pipeline *or* directly to the primary collection, depending on whether the data requires transformation before indexing.
- *Index profiles*
  Send your content to an index profile using the Fusion REST API.

Indexing your content



The *index pipeline* consists of one or more configurable *index pipeline stages*, each performing a different type of transformation on the incoming data. Each connector has a default index pipeline, but you can modify these or create new ones.

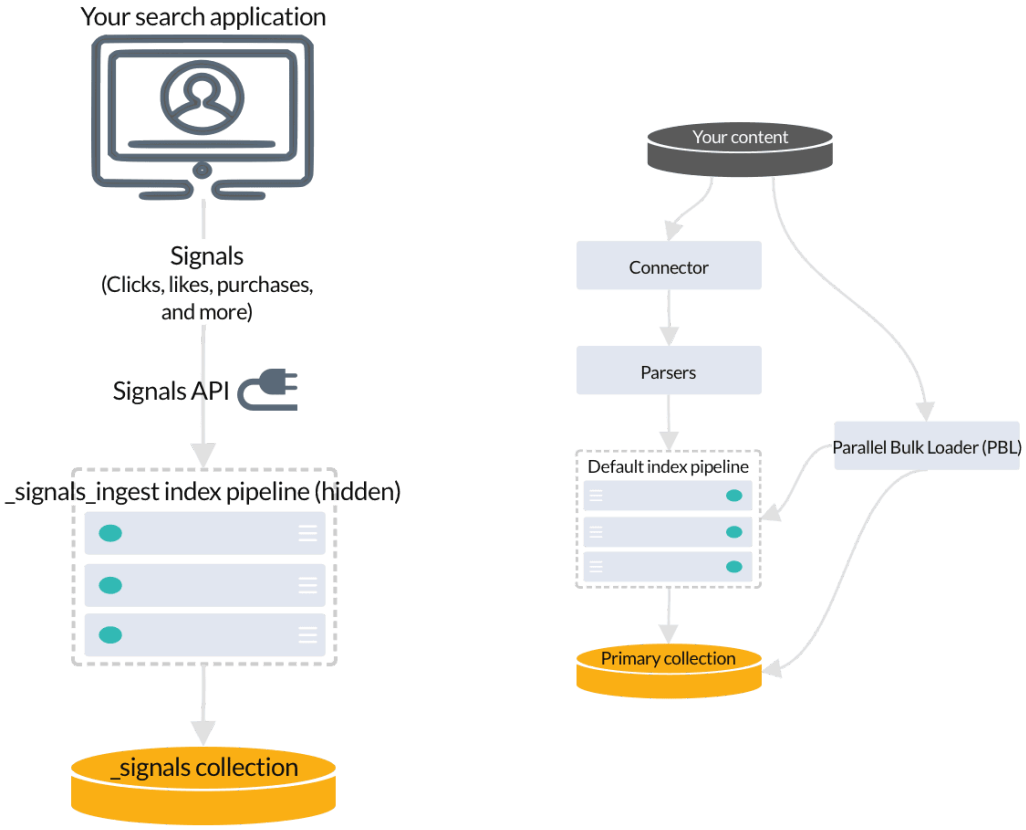
The last stage in any index pipeline should be the *Solr Indexer* stage, which submits the documents to Solr for indexing.

2. Index your signals

Signals are event records that provide historical data about user behavior, such as clicks, likes, purchases, and so on. You don't need to index signals about query responses; Fusion indexes response signals automatically by default. If you are using *App Studio* or *App Insights*, then you need to index request signals. Learn more about *signal types and required fields*.

To index your signals, you send them to Fusion using the *Signals API*, which points to the hidden index pipeline designed especially for signals.

Indexing your signals

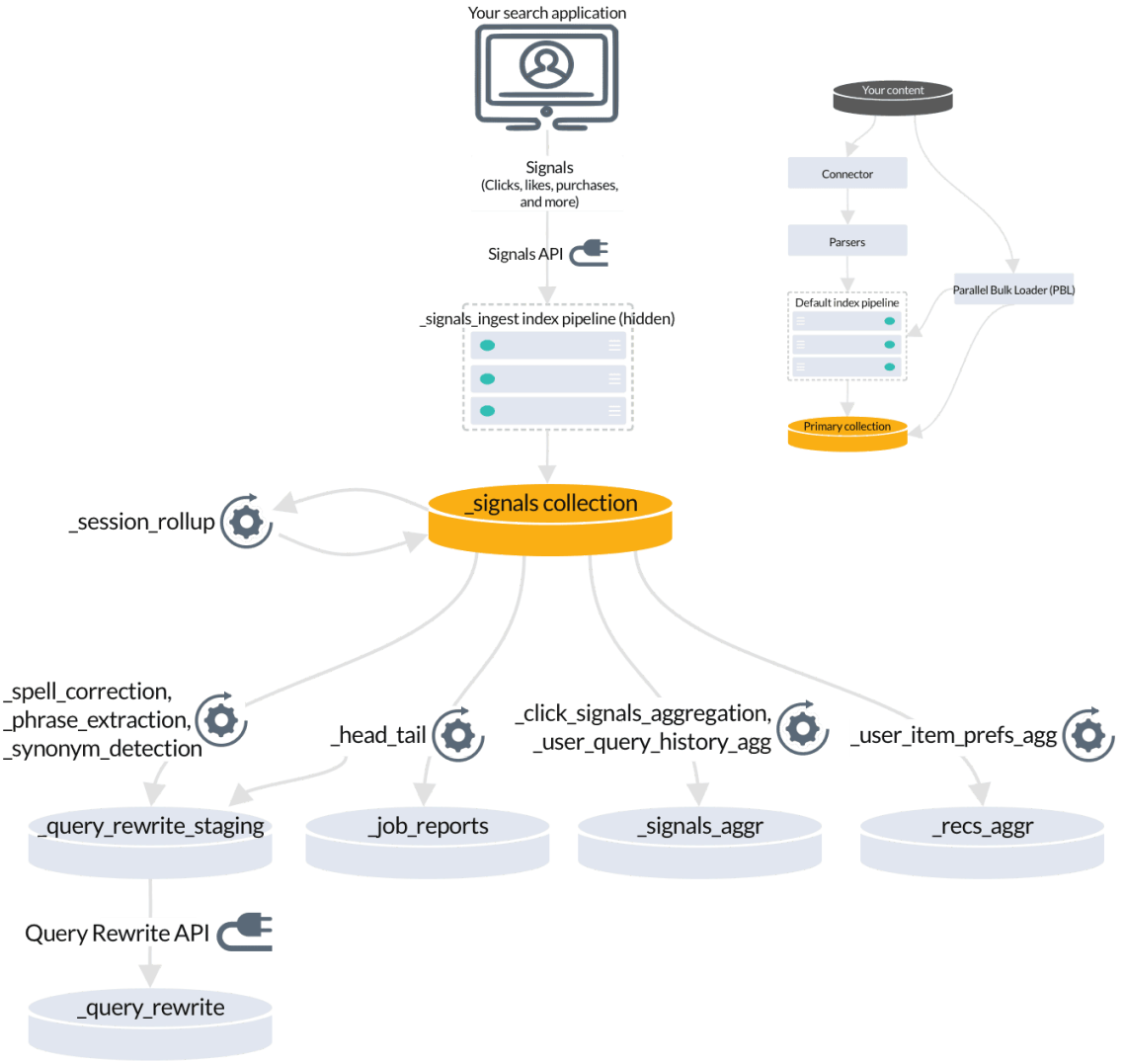


Raw signals are indexed in a secondary collection called `COLLECTION_NAME_signals`. For example, if your primary collection is called `Products`, then the raw signals collection is `Products_signals`.

3. Fusion jobs that index signals-related data

When you *enable signals*, Fusion creates jobs and secondary collections for analyzing and aggregating your raw signals. Some of this data enables *query rewriting* and *automatic boosting*, while other data becomes useful when you enable *recommendations*.

Fusion jobs that index signals-related data

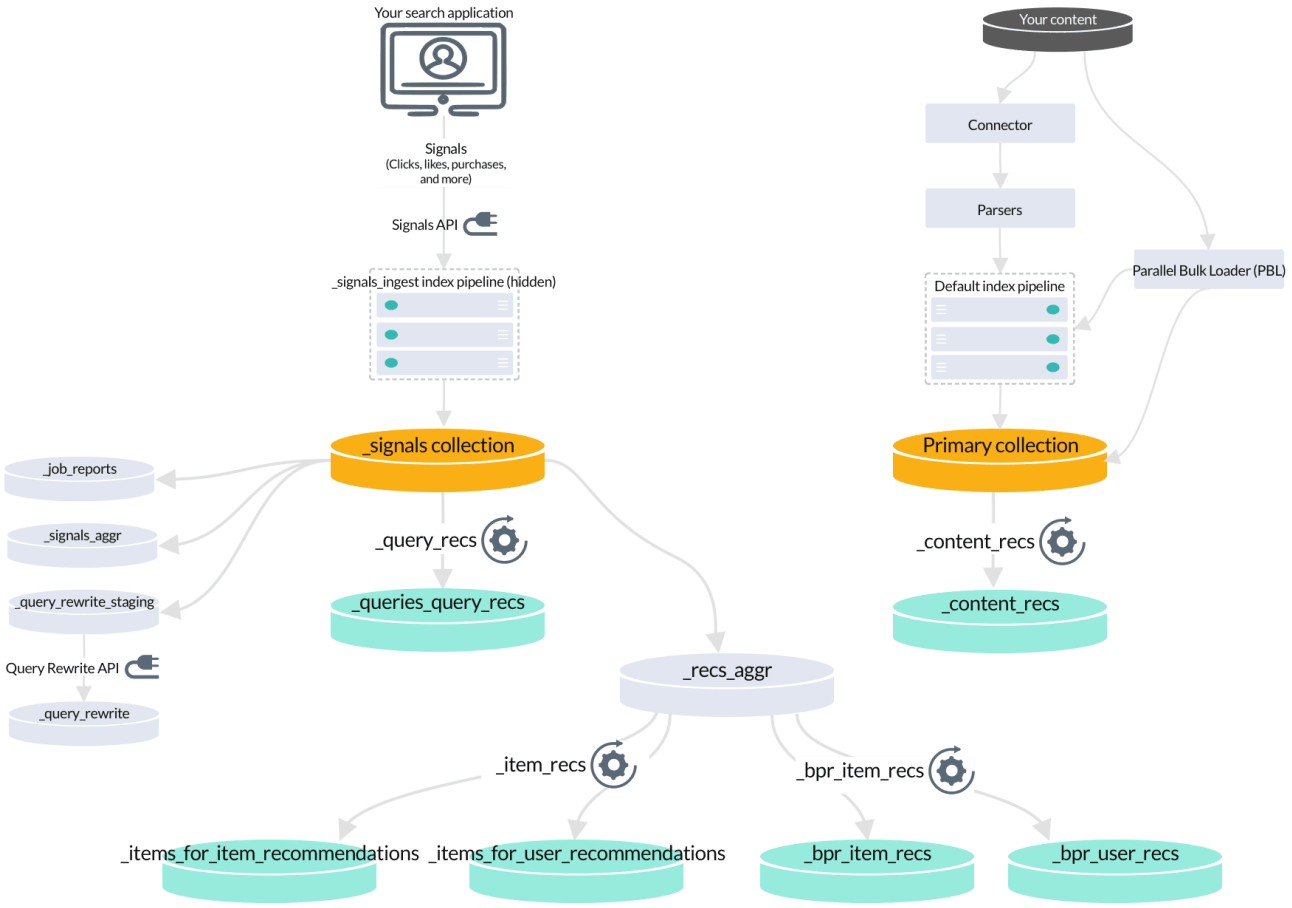


- The `_session_rollup` *job* creates session signals from your raw signals and adds them to the `_signals` collection.
- *Automatic query rewrites* are created by *several jobs* and indexed in the `_query_rewrite_staging` collection for review before they are published to the `_query_rewrite` collection.
- The `_head_tail` *job* sends *rewrites for underperforming queries* to the the `_query_rewrite_staging` collection. It also sends analytics tables to the `_job_reports` collection.
- The `_signals_aggr` collection stores unique combinations of query, document ID, and filters found in your raw signals, each with a timestamp, weight, and count. These are generated by the `_click_signals_aggregation` *job* and the `_user_query_history_agg` *job*.
- The `user_item_prefs_agg` *job* identifies unique pairs of document IDs and user IDs, assigns a timestamp, weight, and count to each pair, and indexes it in the `_recs_aggr` collection.

4. Fusion jobs that index recommendations

When you *enable recommendations*, another set of jobs and secondary collections is created.

Fusion jobs that index recommendations



The default recommendation jobs read data from different collections depending on the type of recommendations being generated:

- *Content-based recommendations* are produced by the `_content_recs` *job* by analyzing your primary collection.
- *Queries-for-query recommendations* are produced by the `_query_recs` *job* by analyzing your raw signals.
- BPR-based *items-for-item* and *items-for-user* recommendations are produced by the `_bpr_item_recs` *job* by analyzing aggregated signals.

	For more recommendations, try *configuring the Trending Recommender job*.

What's next?

See *Query Data Flow* to learn about querying your content and recommendations.

Query Data Flow

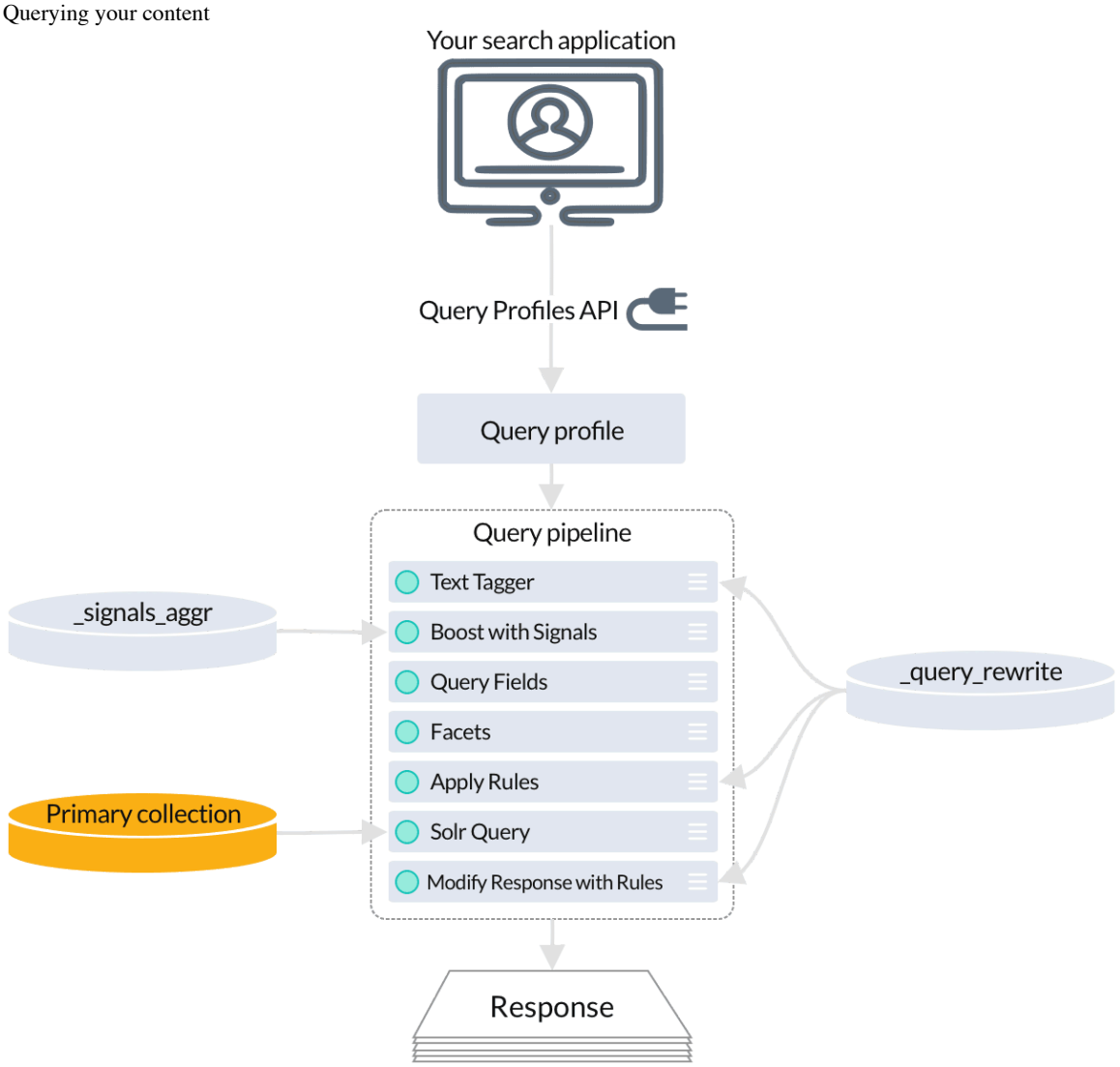
Indexing Data Flow shows you how your content, signals, and recommendations are indexed in Fusion. This topic explains how to query your indexed data.

1. Query your content

When you query your content using the default query pipeline *and* you have *signals enabled*, you also get *query rewrites* and *automatic boosting* to enhance the relevancy of your search results. In addition to matches from your primary collection, you'll get results produced by Fusion's machine learning jobs, based on *signals* from your users.

Query profiles give you a stable endpoint that can be associated with any *query pipeline*. Just specify the query profile when sending queries to the *Query API* endpoint. You can always reconfigure the query profile to point to a different pipeline, or to an *experiment*.

Querying your content

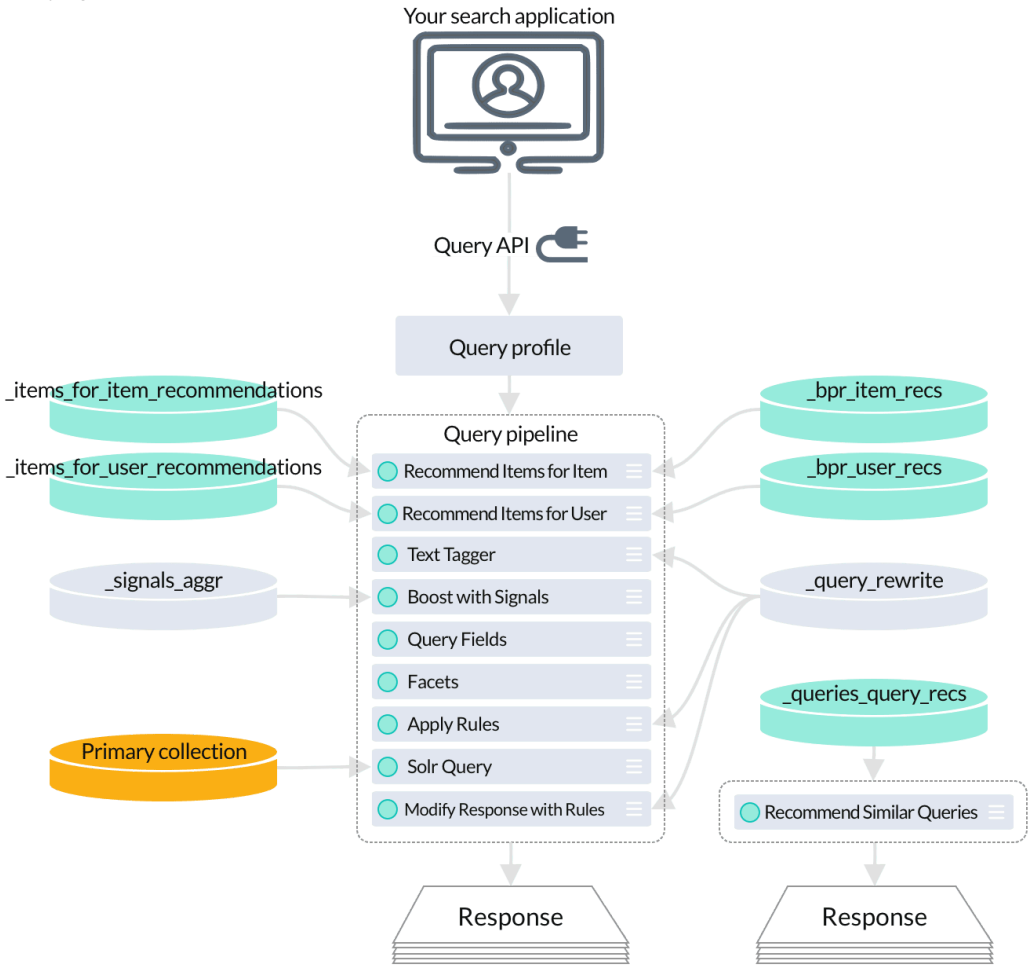


- Several stages perform *query rewriting* and *response rewriting* by reading from the `_query_rewrite` collection:
  - The *Text Tagger stage* finds *misspellings*, *phrases*, *underperforming query strings*, and *synonyms* in the incoming query.
  - The *Apply Rules stage* applies your *business rules* to the incoming query.
  - The *Modify Response with Rules stage*
- The *Boost with Signals stage* reads aggregated signals to boost search results that have been popular with users.
- The *Solr Query stage* fetches the final set of relevant search results from the primary collection.

2. Query for recommendations

To get even more relevant results based on AI-powered analysis of your signals data, your query pipeline can perform any number of secondary queries to the collections where the output from machine learning jobs is indexed. Your secondary queries can provide additional search results based on Fusion's analysis of your signals.

Querying for recommendations



- Use the *Recommend Items for Item stage* to query for *collaborative items-for-item recommendations*.
- The *Recommend Items for User stage* queries for *items-for-user recommendations*.
- The *Recommend Similar Queries stage* provides *queries-for-query recommendations*.

Machine Learning

Machine learning with Spark

Apache Spark *(http://spark.apache.org/)* is an open-source cluster-computing framework that serves as a fast and general execution engine for large-scale data processing jobs that can be decomposed into stepwise tasks, which are distributed across a cluster of networked computers.

Spark improves on previous MapReduce implementations by using resilient distributed datasets (RDDs), a distributed memory abstraction that lets programmers perform in-memory computations on large clusters in a fault-tolerant manner.

Fusion manages a Spark cluster that is used for all *signal aggregation* processes.

With a Fusion license, you can also use the Spark cluster to *train and compile machine learning models*, as well as to run experiments via the *Fusion UI* or the *Spark Jobs API*.

See *Machine Learning Jobs* for details about each pre-defined machine learning job in Fusion.

More information

- Apache Spark Key Terms, Explained *(https://databricks.com/blog/2016/06/22/apache-spark-key-terms-explained.html)*
- Apache Spark on Wikipedia *(https://en.wikipedia.org/wiki/Apache_Spark)*
- *Machine Learning Models in Fusion*
- *Machine Learning Jobs*
- *Spark Administration in Kubernetes*

Machine Learning Jobs

Fusion provides these job types to perform machine learning tasks.

Signals analysis

These jobs analyze a collection of signals in order to perform *query rewriting*, *signals aggregation*, or *experiment analysis*.

- *Ground Truth*
  Estimate ground truth queries using click signals and query signals, with document relevance per query determined using a click/skip formula.

Query rewriting

These jobs produce data that can be used for query rewriting or to inform updates to the *synonyms.txt file*.

- *Head/Tail Analysis*
  Perform head/tail analysis of queries from collections of raw or aggregated signals, to identify underperforming queries and the reasons. This information is valuable for improving overall conversions, Solr configurations, auto-suggest, product catalogs, and SEO/SEM strategies, in order to improve conversion rates.
- *Phrase Extraction*
  Identify multi-word phrases in signals.
- *Synonym Detection Jobs*
  Use this job to generate pairs of synonyms and pairs of similar queries. Two words are considered potential synonyms when they are used in a similar context in similar queries.
- *Token and Phrase Spell Correction*
  Detect misspellings in queries or documents using the numbers of occurrences of words and phrases.

Signals aggregation

- *SQL Aggregation*
  A Spark SQL aggregation job where user-defined parameters are injected into a built-in SQL template at runtime.

Experiment analysis

- *Ranking Metrics*
  Calculate relevance metrics (nDCG and so on) by replaying ground truth queries against catalog data using variants from an experiment.

Collaborative recommenders

These jobs analyze signals and generate matrices used to provide *collaborative recommendations*.

- *BPR Recommender*
  Use this job when you want to compute user recommendations or item similarities using a Bayesian Personalized Ranking (BPR) recommender algorithm.
- *Query-to-Query Session-Based Similarity jobs*
  This recommender is based on co-occurrence of queries in the context of clicked documents and sessions. It is useful when your data shows that users tend to search for similar items in a single search session. This method of generating query-to-query recommendations is faster and more reliable than the Query-to-Query Similarity recommender job, and is session-based unlike the similar queries previously generated as part of the *Synonym Detection job*.

Content-based recommenders

Content-based recommenders create matrices of similar items based on their content.

- *Content-Based Recommender*
  Use this job when you want to compute item similarities based on their content, such as product descriptions.

Content analysis

- *Cluster Labeling*
  Use this job when you already have clusters or well-defined document categories, and you want to discover and attach keywords to see representative words within those existing clusters. (If you want to create new clusters, use the *Document Clustering job*.)

- *Document Clustering*
  The Document Clustering job uses an unsupervised machine learning algorithm to group documents into clusters based on similarities in their content. You can enable more efficient document exploration by using these clusters as facets, high-level summaries or themes, or to recommend other documents from the same cluster. The job can automatically group similar documents in all kinds of content, such as clinical trials, legal documents, book reviews, blogs, scientific papers, and products.
- *Classification job*
  This job analyzes how your existing documents are categorized and produces a classification model that can be used to predict the categories of new documents at index time.
- *Outlier Detection*
  Use this job when you want to find outliers from a set of documents and attach labels for each outlier group.

Data ingest

- *Parallel Bulk Loader*
  The Parallel Bulk Loader (PBL) job enables bulk ingestion of structured and semi-structured data from big data systems, NoSQL databases, and common file formats like Parquet and Avro.

Machine Learning Models in Fusion

For information about machine learning models, see *Develop and Deploy a Machine Learning Model*.

Natural Language Processing

This topic describes Fusion's Natural Language Processing (NLP) features, available in the legacy OpenNLP NER Extraction index pipeline stage and NLP Annotator index and query pipeline stages.

OpenNLP NER Extraction pipeline stage

The *OpenNLP NER Extraction index pipeline stage* performs only *Named Entity Recognition (NER)*. This stage is available in all versions of Fusion.

For additional NLP functionality, use the NLP Annotator pipeline stages, available in Fusion versions 4.2 and later. See below for details.

NLP Annotator pipeline stages

The NLP Annotator is both an *index pipeline stage* and a *query pipeline stage*. The NLP Annotator performs a variety of fundamental NLP tasks:

- *Sentence detection*
- *Named Entity Extraction (NER)*
- *Part-of-Speech (POS) Tagging*

If configured in an index pipeline, the NLP annotator performs selected NLP tasks on raw document content during the indexing process (see more details here). If configured in a query pipeline, the NLP annotator performs selected NLP tasks on the query text content (see more details here).

NLP features

Fusion's NLP Annotator pipeline stages include the NLP features described below.

Sentence detection

Sentence detection is the process of analyzing text to determine sentence boundaries. It is typically the first step taken when performing any kind of natural language processing on a document. Commonly, a sentence is indexed as a multi-value field that can be used for various purposes, as in these examples:

- Relevancy: Boost documents whose first sentence matches the query terms.
- Snippets: When presenting the search results, display the first few sentences of each document.

Named Entity Recognition (NER)

Named Entity Recognition is a popular technique used in information extraction to identify and segment the named entities and classify or categorize them under these predefined classes:

- person
- organization
- location

For example:

Person		**Organization**		**Location**
Jane	is the CEO of	Example Company	, based in	San Francisco.

Name entity recognition is widely leveraged by today's text mining projects. When organizations store large volumes of business documents in Fusion, the natural next step is to turn the large volume of text-centric data into some kind of knowledge base.

Take entity linking projects, for example: The client may want to link all relevant documents with an existing list of entities of interest. One way of doing this is to extract entities from all raw text documents, then perform fuzzy matching or another kind of text pattern matching to link relevant documents with a specific entity from the given list. This is more efficient than scanning the whole document and trying to search for the entity name. In this scenario, NER extraction is an ideal tool.

Fusion has integrated NER capability into its indexing and query pipelines to enable customers to perform knowledge discovery easily.

Part-of-Speech (POS) tagging

One of the most important roles of POS tagging is "word sense disambiguation". For instance, when searching for the word "present", if the intent is to look for the concept of gift, then having the word "present" tagged as a "noun" will help filter out content with "present" as a verb, representing an action of bringing before the public.

Getting Data In

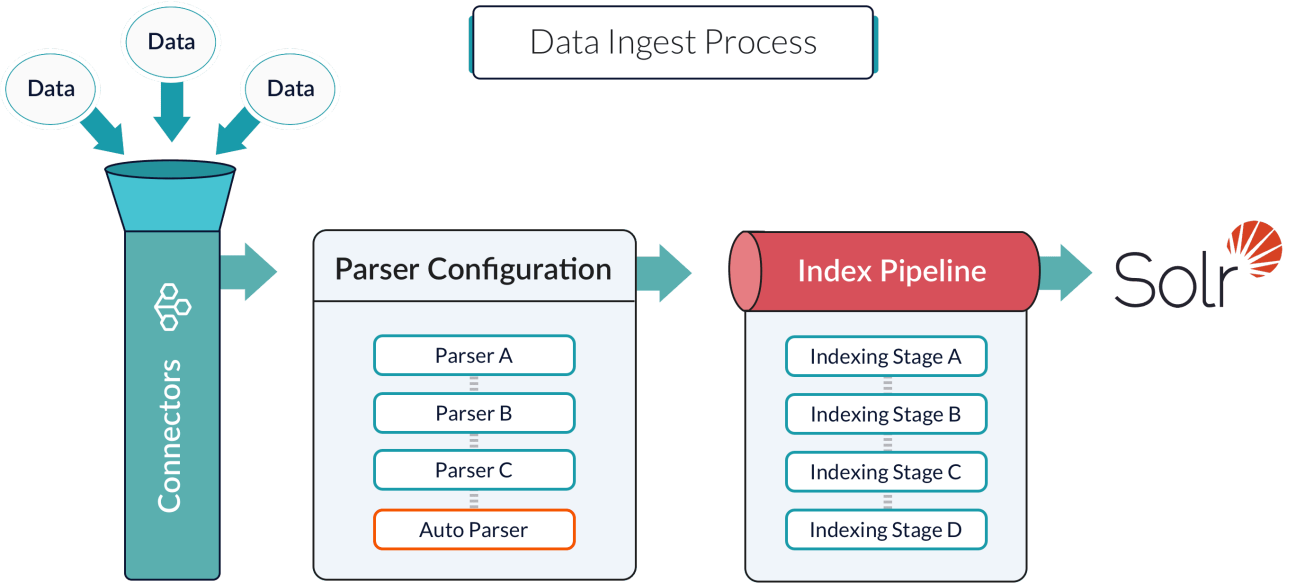
There are many methods for getting data into Fusion, depending on the type of data:

- Your searchable content, *signals*, and logs are all ingested by *indexing them* in Solr.
- *connectors*, *machine learning models*, *JDBC drivers*, *analytics catalogs*, and *Banana dashboards* can be uploaded to the blob store.
- The *Parallel Bulk Loader (PBL)* can perform distributed reads on any data source that implements the SparkSQL (2.2.1 or later) Data Sources API.
- Though signals are normally indexed, you can also *import them*.
- In some cases, you might find that it's best to use other ingestion methods, such as *Hive*, *Pig*, or pushing data to a *REST API* endpoint.

Indexing Your Data

Indexing stores your data in a format that is optimized for searching. How you index your data is critical for ensuring that your data is stored in a format that is ideal for your search application.

- *Index pipelines* determine the details of the conversion.
- *Index pipeline stages* are the building blocks of index pipelines.
- The *Index Workbench* is Fusion's index pipeline development tool.



Collections

Collections are a way of grouping data sets so that related data sets can be managed together. Every data set that you ingest belongs to a collection. Any app can contain one or more collections. See *Collection Management*.

Datasources

Datasources are configurations that determine how your data is handled during ingest by Fusion's connectors, parsers, and index pipelines. When you run a fully-configured datasource, the result is an indexed data set that is optimized for search, depending on the shape of your data and how you want to search it. See *Configure a New Datasource*.

	For additional guidance on configuring datasouces, please visit our Indexing Data *(https://academy.lucidworks.com/indexing-data)* training module.

Connectors

Connectors are Fusion components that ingest and parse specific kinds of data. There is a Fusion *connector* for just about any data type.

Additional resources

- View our training on Indexing Data *(https://academy.lucidworks.com/indexing-data)*.

Indexing Data Flow

With Fusion, your data is indexed in a set of collections in Fusion's Solr core.

A *primary collection* holds your searchable content, such as your product catalog, knowledge base, blog articles, product reviews, and so on.

A set of *secondary collections* are associated with your primary collection to hold related data that Fusion can use to enhance the relevancy of your search results.

This topic shows you how different types of data flow through Fusion to be indexed in your primary and secondary collections.

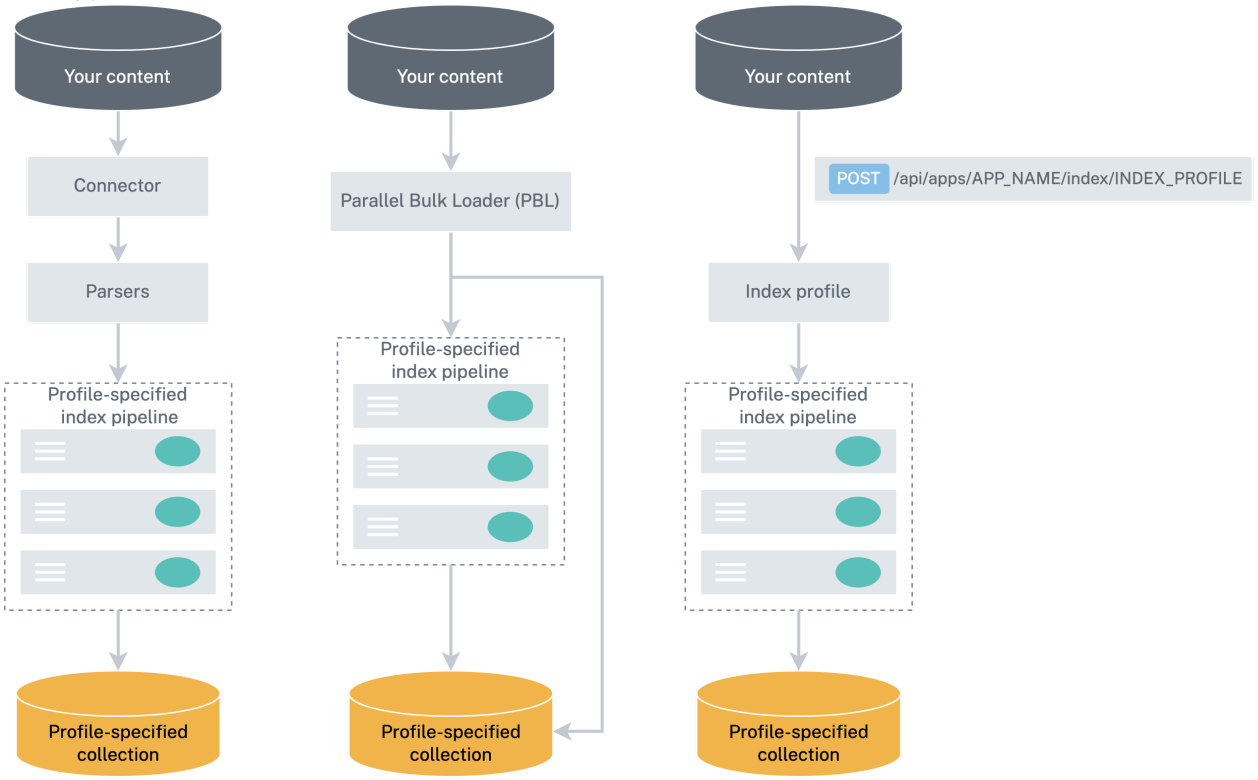
1. Index your content

No matter where your content is located, Fusion can index it. The Fusion collection where your searchable content is indexed is called the primary collection. Secondary collections are automatically created to hold related data, such as signals, machine learning job output, and so on.

There are a few ways to index your searchable content to the primary collection:

- *Connectors*
  Lucidworks has a wide variety of connectors for many types of data sources. *Find your connector*.
  Once the connector fetches your data, *parsers* read it before passing it to the *index pipeline*.
- The *Parallel Bulk Loader (PBL)*
  The PBL can send your data to an index pipeline *or* directly to the primary collection, depending on whether the data requires transformation before indexing.
- *Index profiles*
  Send your content to an index profile using the Fusion REST API.

Indexing your content



The *index pipeline* consists of one or more configurable *index pipeline stages*, each performing a different type of transformation on the incoming data. Each connector has a default index pipeline, but you can modify these or create new ones.

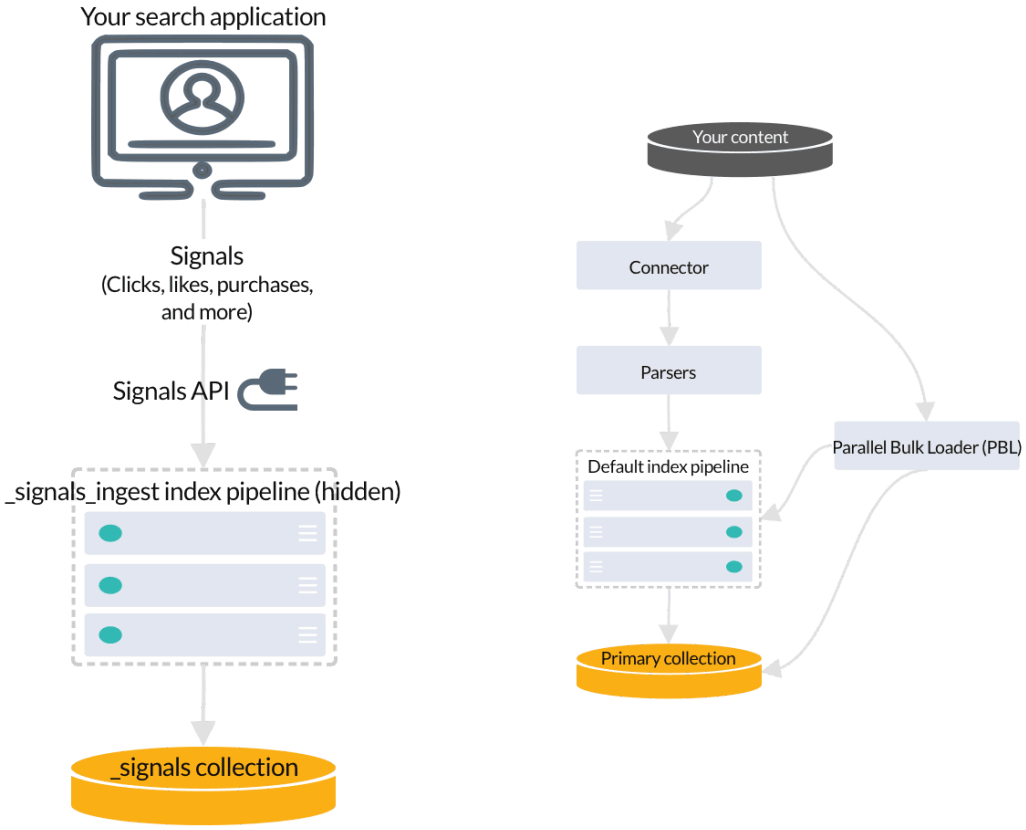
The last stage in any index pipeline should be the *Solr Indexer* stage, which submits the documents to Solr for indexing.

2. Index your signals

Signals are event records that provide historical data about user behavior, such as clicks, likes, purchases, and so on. You don't need to index signals about query responses; Fusion indexes response signals automatically by default. If you are using *App Studio* or *App Insights*, then you need to index request signals. Learn more about *signal types and required fields*.

To index your signals, you send them to Fusion using the *Signals API*, which points to the hidden index pipeline designed especially for signals.

Indexing your signals

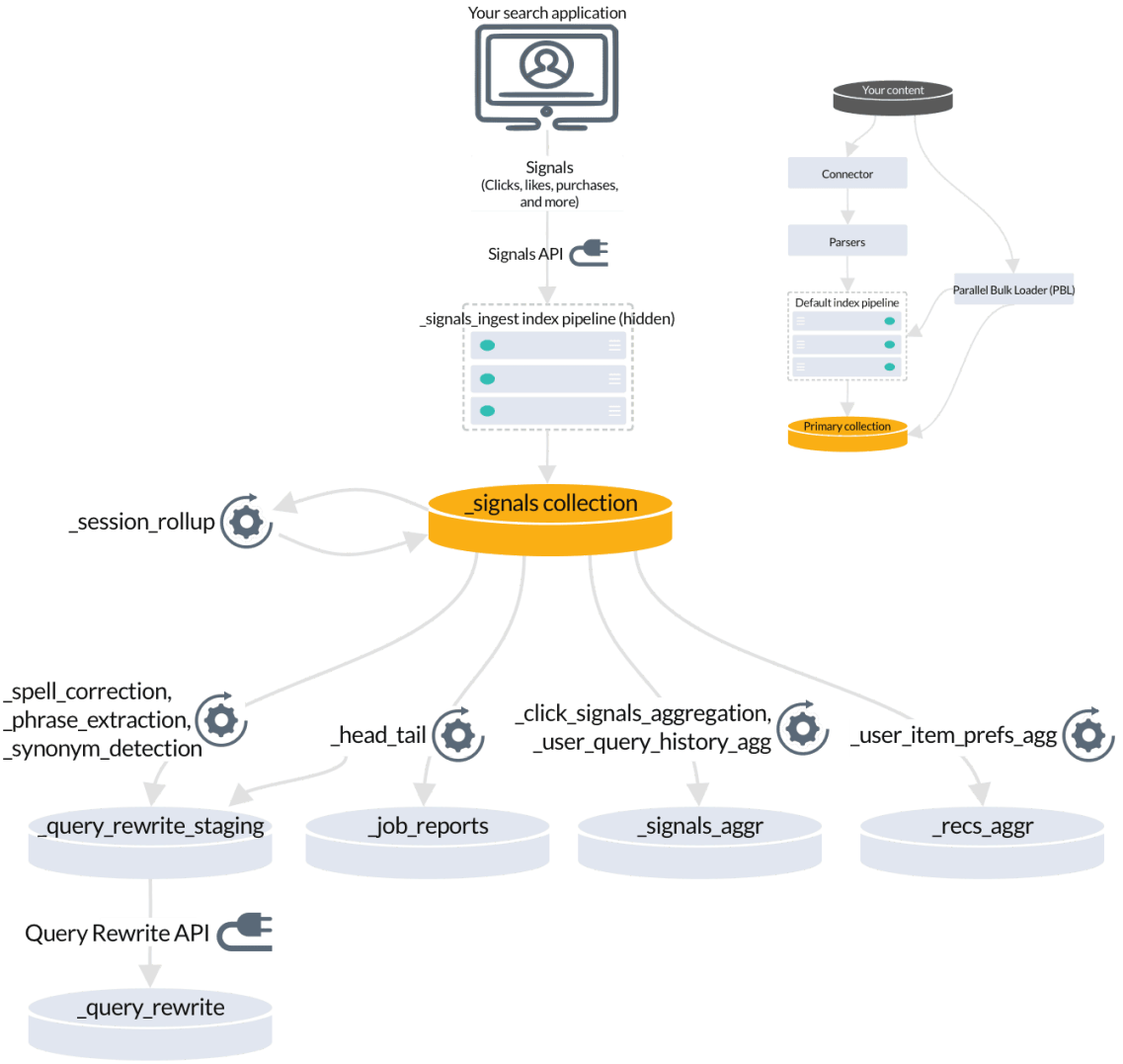


Raw signals are indexed in a secondary collection called `COLLECTION_NAME_signals`. For example, if your primary collection is called `Products`, then the raw signals collection is `Products_signals`.

3. Fusion jobs that index signals-related data

When you *enable signals*, Fusion creates jobs and secondary collections for analyzing and aggregating your raw signals. Some of this data enables *query rewriting* and *automatic boosting*, while other data becomes useful when you enable *recommendations*.

Fusion jobs that index signals-related data

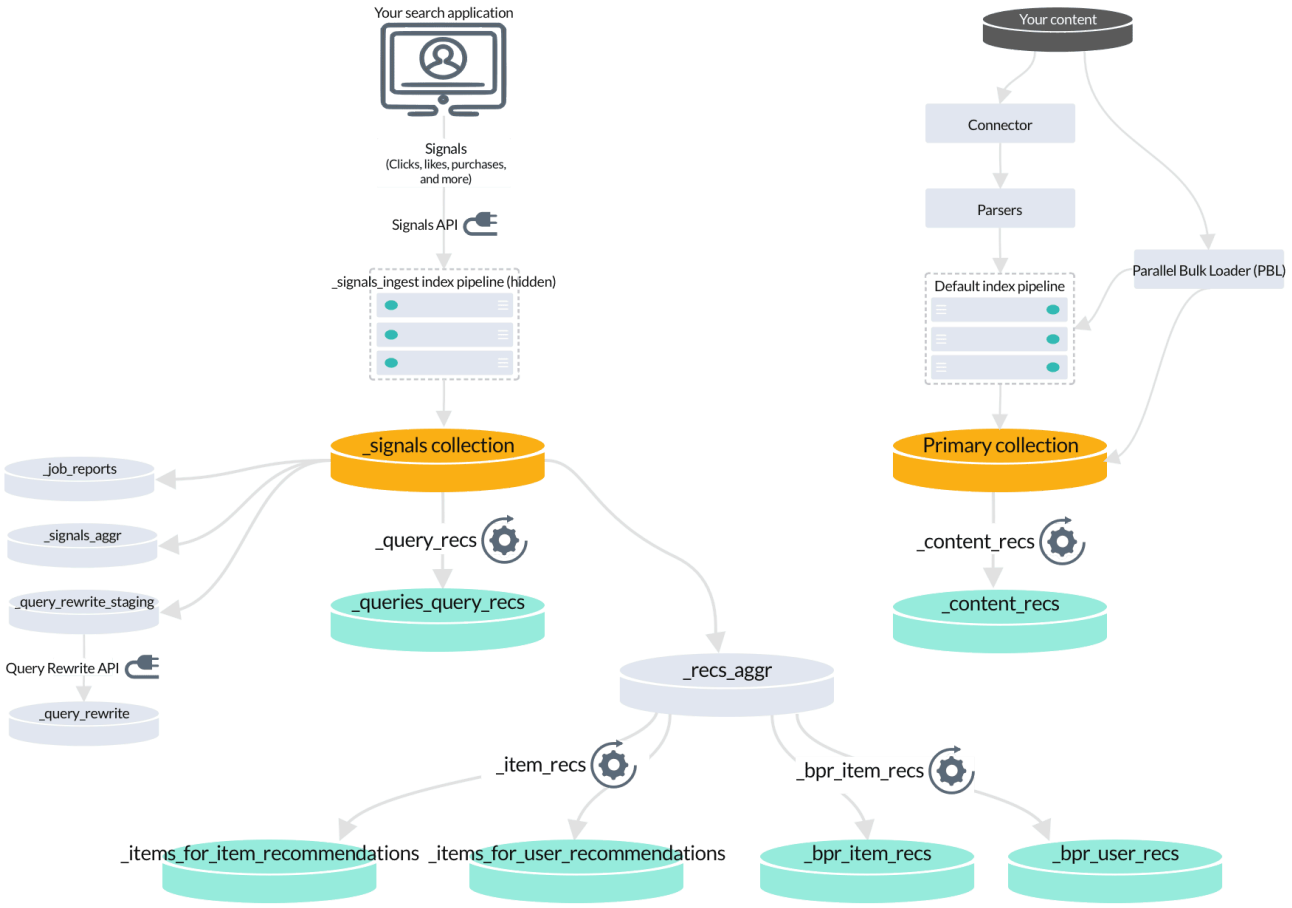


- The `_session_rollup` *job* creates session signals from your raw signals and adds them to the `_signals` collection.
- *Automatic query rewrites* are created by *several jobs* and indexed in the `_query_rewrite_staging` collection for review before they are published to the `_query_rewrite` collection.
- The `_head_tail` *job* sends *rewrites for underperforming queries* to the the `_query_rewrite_staging` collection. It also sends analytics tables to the `_job_reports` collection.
- The `_signals_aggr` collection stores unique combinations of query, document ID, and filters found in your raw signals, each with a timestamp, weight, and count. These are generated by the `_click_signals_aggregation` *job* and the `_user_query_history_agg` *job*.
- The `user_item_prefs_agg` *job* identifies unique pairs of document IDs and user IDs, assigns a timestamp, weight, and count to each pair, and indexes it in the `_recs_aggr` collection.

4. Fusion jobs that index recommendations

When you *enable recommendations*, another set of jobs and secondary collections is created.

Fusion jobs that index recommendations



The default recommendation jobs read data from different collections depending on the type of recommendations being generated:

- *Content-based recommendations* are produced by the `_content_recs` *job* by analyzing your primary collection.
- *Queries-for-query recommendations* are produced by the `_query_recs` *job* by analyzing your raw signals.
- BPR-based *items-for-item* and *items-for-user* recommendations are produced by the `_bpr_item_recs` *job* by analyzing aggregated signals.

	For more recommendations, try *configuring the Trending Recommender job*.

What's next?

See *Query Data Flow* to learn about querying your content and recommendations.

Collections

Your data is organized into collections. When you create an app, Fusion automatically creates a collection with the same name. You can create additional collections in any app.

A primary collection contains the data that your users will search. Every primary collection is associated with a set of auxiliary collections that contain related data, such as signals, aggregations, and more.

Under the hood, a Fusion collection is a distributed index in Solr, defined by a named configuration stored in ZooKeeper, with these properties:

- **Number of shards.** Documents are distributed across this number of partitions.
- **Document routing strategy.** How documents are assigned to shards.
- **Replication factor.** How many copies of each document in the collection.
- **Replica placement strategy.** Where to place replicas in the cluster.

If your data is already stored in a Solr instance or cluster, you can manage this collection in Fusion by creating a Fusion collection that imports the existing Solr collection.

	Collection names are case-insensitive, but Fusion preserves case when displaying collection names.

Auxiliary Collections

Every primary collection is associated with a set of auxiliary collections that contain related data, such as signals, aggregations, and more.

Some auxiliary collections are created for every primary collection. Others are created only for the app's default collection, one per app.

Auxiliary collections are described below:

`APP_NAME_job_reports`	Output from Fusion *experiments*, *Ranking Metrics jobs*, and *Head/Tail Analysis jobs*.	1 per app
`APP_NAME_query_rewrite`	A collection of documents to use for *rewriting queries*, optimized for high-volume traffic. These documents originate from the `COLLECTION_NAME_query_rewrite_staging` collection. Certain Fusion query pipeline stages read from this collection: • *Text Tagger* • *Apply Rules* • *Modify Response with Rules*	1 per app
`APP_NAME_query_rewrite_staging`	A collection of documents created by the Rules Editor or by certain *Fusion jobs*, not optimized for production traffic. Documents move from this collection to the `COLLECTION_NAME_query_rewrite` collection as follows: • Job output documents with high confidence contain a `review=auto` field and are moved to the `COLLECTION_NAME_query_rewrite` collection automatically. • Job output documents with low confidence contain a `review=pending` field. When these are approved by a Fusion user, Fusion copies them to the `COLLECTION_NAME_query_rewrite` collection.	1 per app
`COLLECTION_NAME_signals`	A search query logs and signals collection.	1 per collection
`COLLECTION_NAME_signals_aggr`	A collection for aggregated signals.	1 per collection
`APP_NAME_user_prefs`	A collection of data to support App Studio's social features, such as user-generated tags, bookmarks, comments, ratings, and so on.	1 per app

	Don’t create primary collections with names that end in the suffixes above; these are reserved for Fusion auxiliary collections, which are created and managed by Fusion directly.

Fusion maintains a set of Solr collections that store Fusion’s own log files and other internal information. These are called *System Collections*, described below.

	Don’t create primary collections named "logs" or beginning with "system_". These names are reserved for Fusion system collections.

Fusion uses ZooKeeper to register information about all collections, and the Fusion components and services related to a collection. The Fusion components associated with a collection include:

- Datasources
- Pipelines
- Profiles
- Signals and aggregations
- Analytics dashboards

System Collections

Fusion automatically creates some collections that are used for internal purposes and shared across all apps:

- **system_autocomplete** stores the content that the Fusion UI displays when you use the search bar.
- **system_blobs** stores *blobs* in Solr. This is used to store model files for the NLP components and other binary files used by Fusion components.
- **system_history** keeps a record of configuration changes, start and stop times for services and experiments, and more.
- **system_jobs_history** keeps a record of Fusion *jobs*, including start/stop times and status.
- **system_logs** stores parsed Java logs from the REST API, connectors-classic component, and other parts of Fusion, like proxy and connectors-rpcs. It also includes http logs and optional gc logs (off by default).

Collection Configuration Properties

Collections have three properties that you can configure only when you are creating a collection using the *Collections API*.

Property	Description	Default behavior
signals *	The `signals` property determines whether to create auxiliary collections with suffixes `_signals` and `_signals_aggr`.	When you create a collection in the Fusion UI, `signals` defaults to **true**. When you create a collection using the Fusion API, this property defaults to **false**.
searchLogs	The `searchLogs` property determines whether to create an auxiliary search query logs collection with suffix `_logs`.	When you create a collection in the Fusion UI, this property defaults to **true**. When you create a collection using the Fusion API, this property defaults to **false**.

*Signals are events with timestamps that can be used to improve search results. For more information about signals in Fusion, see *Signals* in the Fusion documentation.

In schemaless mode, if a document contains a field not currently in the Solr schema, Solr processes the field value to determine what the field type should be defined as, and then adds a new field to the schema with the field name and field type. This behavior can be convenient during preliminary application development, but it’s rarely appropriate in a production environment.

Using profiles to associate collections with pipelines

Index pipelines and query pipelines aren’t connected to a specific collection by default. Index profiles and query profiles are configurations that create consistent endpoints for indexing and querying, each with a specific pipeline and collection.

- *Index Profiles* work with index pipelines for getting content into the system.
- *Query Profiles* work with query pipelines for user queries.

Field Editor UI

The Fields Editor UI allows you to create and configure the schema file directly from Fusion. For instructions, see *Fields Editor UI*.

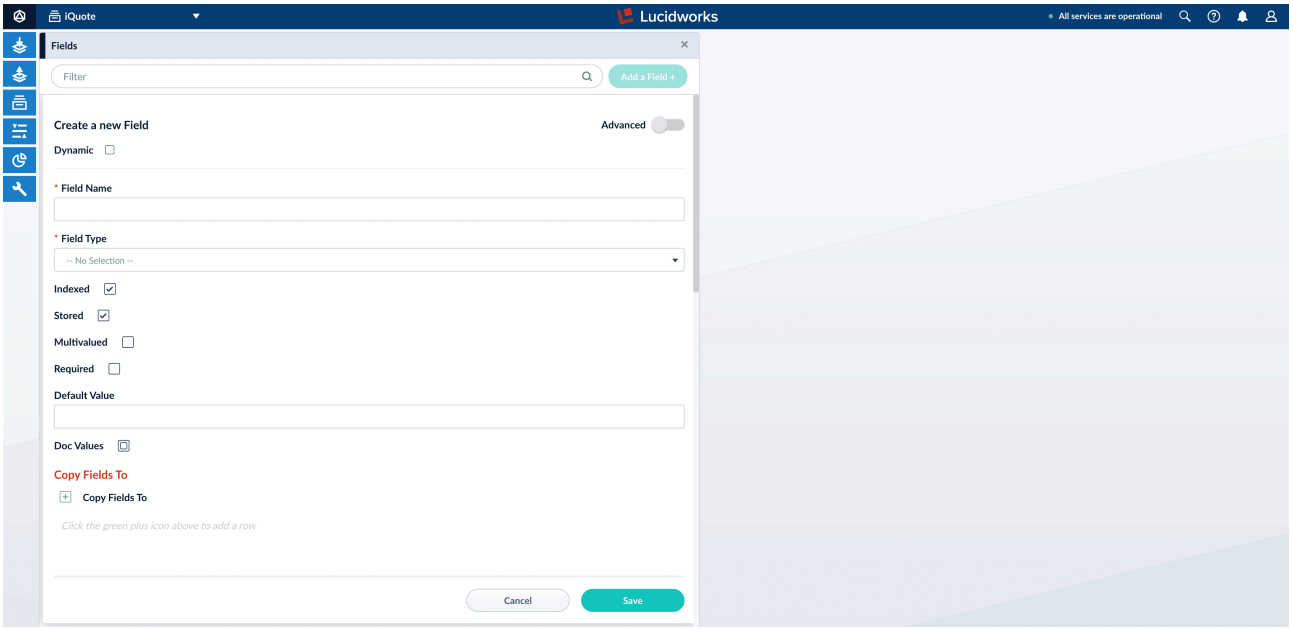
Additional resources

- View our training on Fusion Applications and Collections *(https://academy.lucidworks.com/fusion-applications-and-collections/1706349/scorm/vqpuxkwtzgm1)*.

Fields Editor UI

The Fusion UI includes a space under Collections to edit Fields. Fields help build the schema file and are modified from Fusion with the Fields Editor UI. These fields define how Solr should interpret data in a given field and how to query the fields. You can find descriptions for these fields in the Field Type Definitions section of the Solr Reference Guide *(https://solr.apache.org/guide/8_9/field-type-definitions-and-properties.html)* associated with your Fusion release.

To use the Field Editor, navigate to **Collections** > **Fields** and click **Add a Field+**.

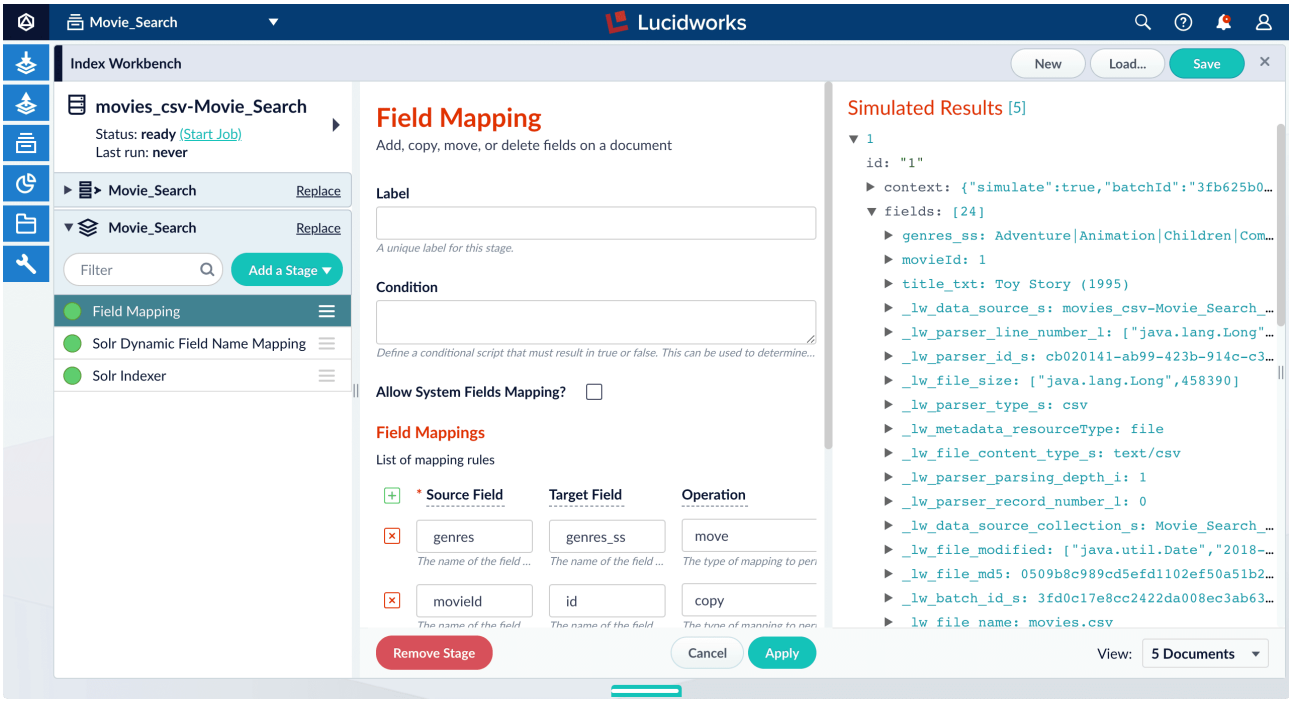


Field options displayed in the UI include:

Option	Input	Notes
Dynamic	checkbox	Fields that are indexed even if they weren't included in the schema.
Field Name	alphanumeric	
Field Type	menu drop-down	
Indexed	checkbox	
Stored	checkbox	
Multi-valued	checkbox	
Required	checkbox	
Default Value	text	
Copy Fields	uses the plus sign to add rows	**static** can copy to `raw_content` or `text`. **dynamic** can copy to any `raw_content`, `text`, or any other dynamic field.
	Advanced toggles checkboxes for **Doc Values**, **Omit Norms**, **Omit Positions**, **Omit term freq and positions**, **Term Vectors**, **Term Positions**, and **Term Offsets**.	

The Indexing Workbench

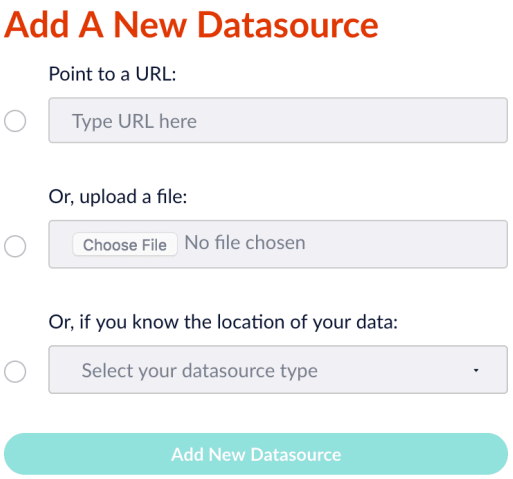
The Index Workbench is a powerful tool that combines key aspects of the data indexing configuration process into one user-friendly panel. It guides the user through the workflow for configuring datasources, parsers and index pipelines, and then running a datasource job to index your data.



Datasources

A datasource is a complete data ingest configuration that consists of the following components:

- **Connector.** Connectors are out-of-the-box components for pulling your data into Fusion. Lucidworks provides a wide variety of connectors, each specialized for a particular data type. For a list of connectors, see *Fusion Connectors*.
- **Parser.** Parsers provide fine-grained configuration for inbound data. See *Parsers* for more information.
- **Index pipeline.** Index pipelines transform incoming data for indexing by Fusion. See *Index Pipelines* for more information.

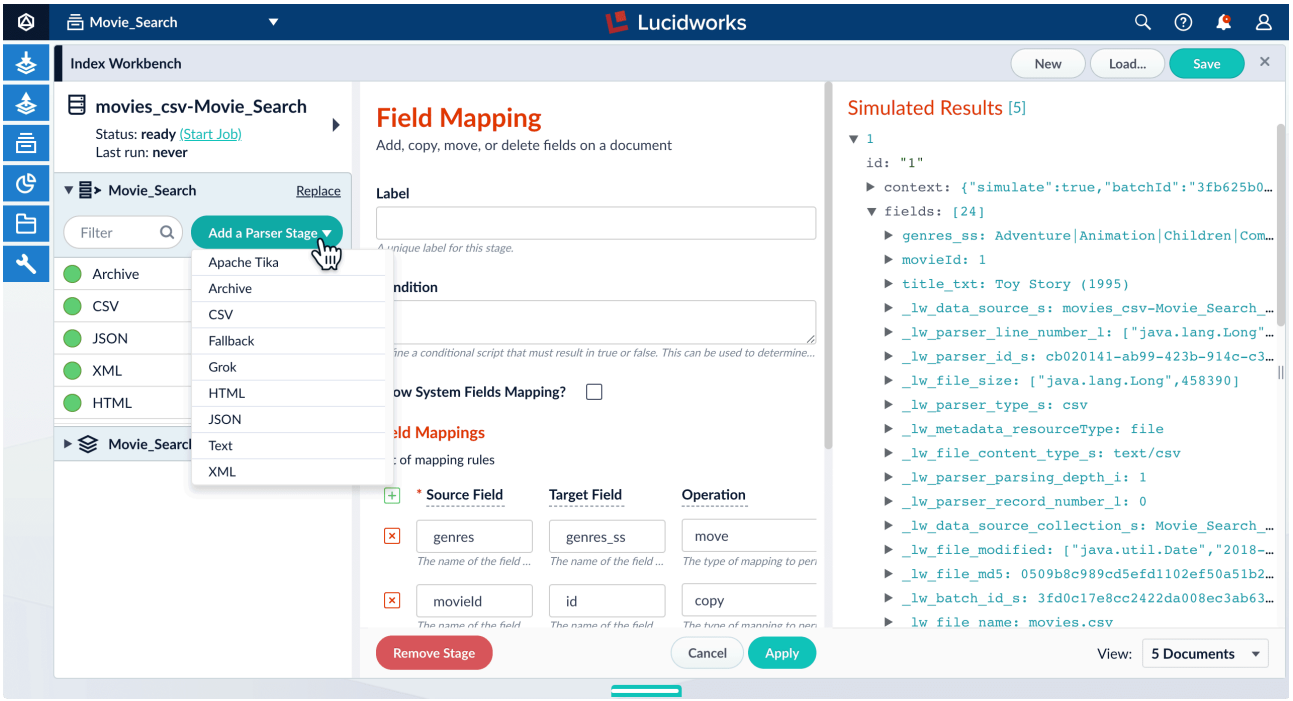


To set up a new datasource in the Index Workbench, click **New**. The **Add A New Datasource** panel appears, offering three simplified methods of configuring how Fusion fetches your data:

- By URL
- By local file upload
- By selecting an installed connector to configure
  See *Connectors Configuration Reference* for a complete list of available connectors. Some are installed by default, and others require separate download and installation.

Parsers

Parsers are a configurable component of the indexing workflow, for flexibility and specificity when parsing inbound data.



A parser consists of an ordered list of parser stages that is completely customizable. The same parser stage can be added to a given configuration multiple times if the different specified settings within those stages best suits the parsing of the data. There is no limit to the number of parser stages that can be included in a parser, and the order in which they run is also completely flexible.

In a parser, after all of the doctype-specific parser stages have run, the Tika and Fallback stages are useful catch-all stages that can attempt to parse anything that has not yet been matched. Tika is used for parsing many types of unstructured documents like PDFs, DOCX, and many more. If all of the other stages in the parser fail to completely parse the data, the Fallback stage can copy the raw bytes directly to Solr.

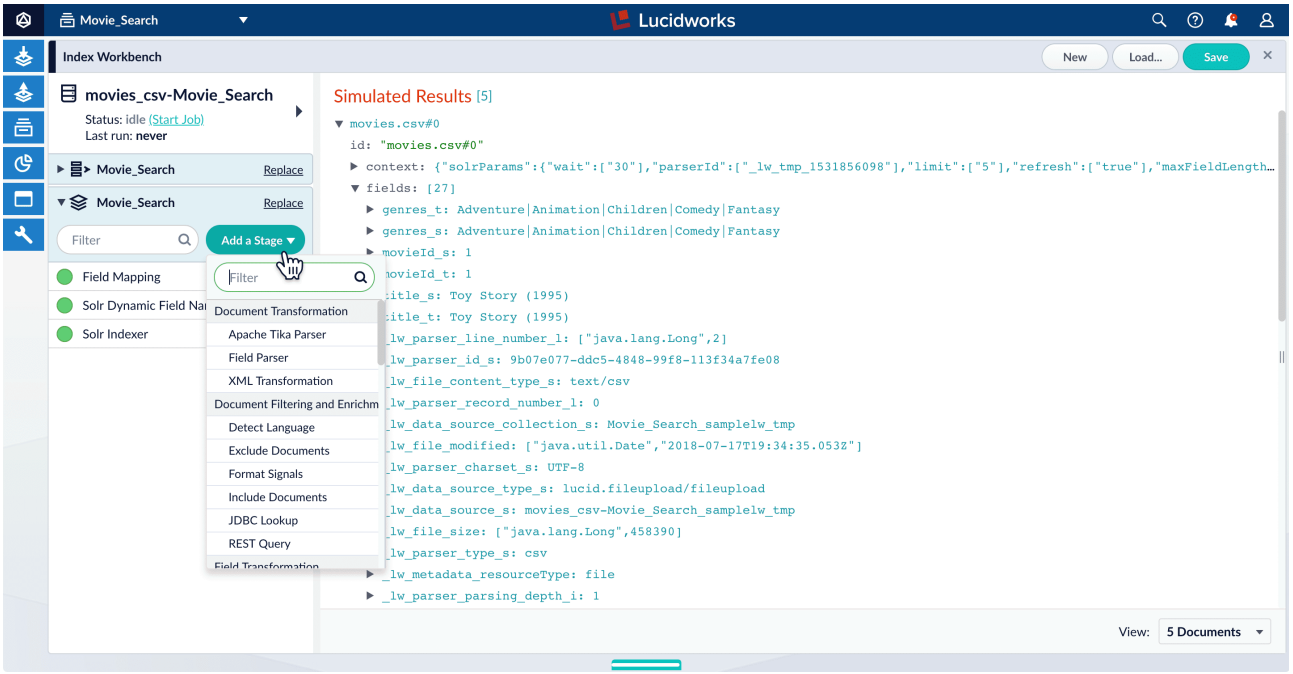
Index pipelines

An *Index Pipeline* transforms incoming data into a document suitable for indexing by Solr via a series of modularized operations called stages. Fusion provides a variety of specialized index stages to index data effectively. Stages can be selected, configured, and enabled or disabled in the Index Pipeline section of the Index Workbench.

Once you finish configuring a datasource using the Index Workbench, you can move on to setting up queries using the *Query Workbench*, which provides a similar workflow for configuring and previewing search results.

Adding a pipeline stage

Click **Add a Stage** to add index pipeline stages that can perform data filtering, transformations, and more.



The *Solr Indexer stage* is the only pipeline stage that is required to complete the indexing process, and therefore exists in every query pipeline. It is always the last stage in a series.

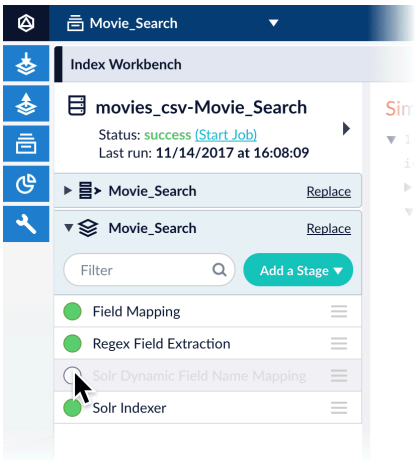
Re-ordering pipeline stages

The order of the pipeline stages matters, because the output from one stage becomes the input to the next stage. For example, the *Solr Indexer stage* must always come last in the sequence, so that data is indexed only after it has been processed by all other stages. Putting this stage first in the sequence means that subsequent stages have no effect on the indexed data.

Click and drag any stage in the pipeline to move it up or down in the sequence of stages. The preview panel automatically updates the results to reflect the output of the new sequence.

Enabling and disabling pipeline stages

By default, every stage in an index pipeline is enabled. While working with a pipeline, it can be helpful to disable stages without removing them completely. This allows you to preserve a stage's configuration while observing how the search results change in its absence. You can re-enable the stage at any time. When you save a query pipeline, the enabled/disabled state of each stage is also saved.

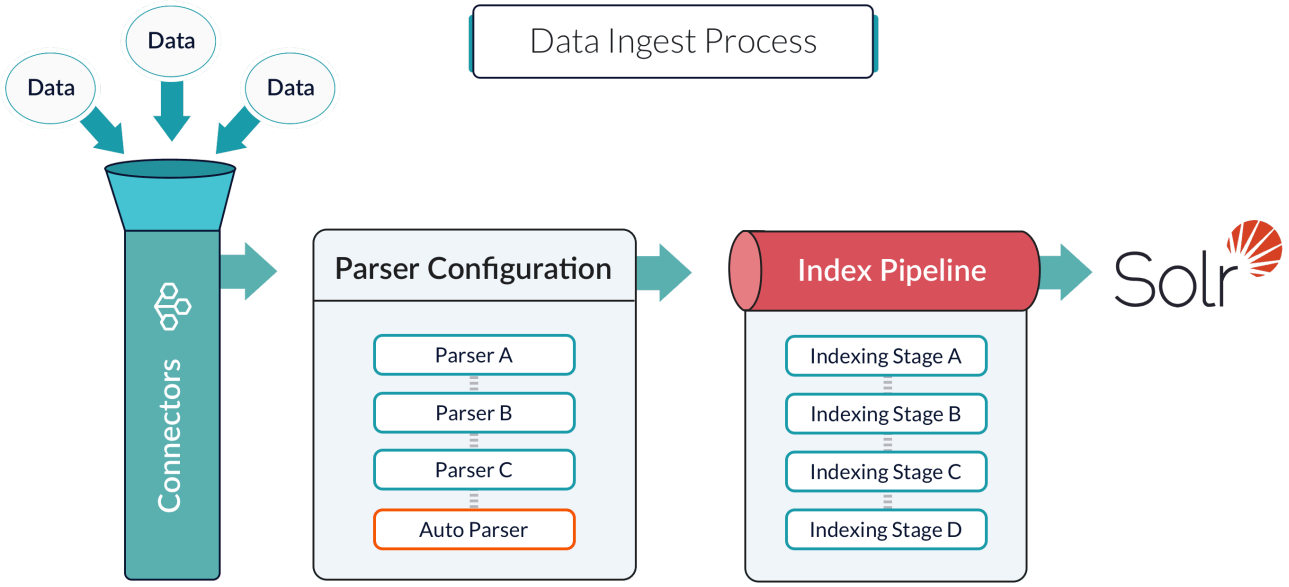


Click the circle next to any stage in order to enable or disable it.

Index Pipelines

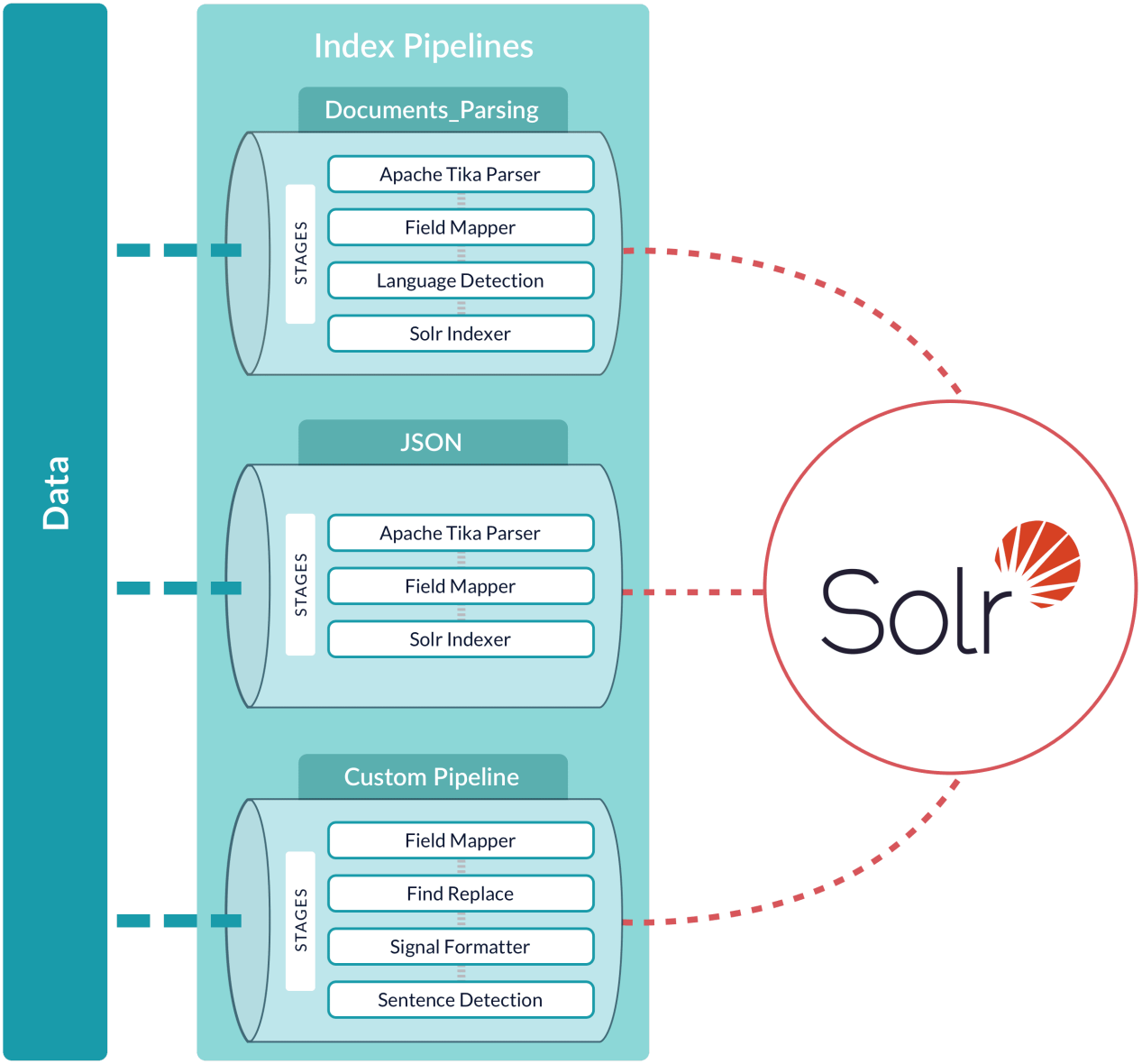
Index pipelines transform incoming data into `PipelineDocument` objects for indexing by Fusion-managed Solr service. An index pipeline consists of a series of configurable *index pipeline stages*, each performing a different transformation on the data before passing the result to the next stage in the pipeline. The final stage is the *Solr Indexer stage*, which transforms the PipelineDocument into a Solr document and submits it to Solr for indexing in a specific *Collection*.

Each configured datasource has an associated index pipeline and uses a *connector* to fetch data to parse and then input into the index pipeline.



Alternatively, documents can be submitted directly to an index pipeline or profile with the REST API; see *Importing Data with the REST API*.

A pipeline can be reused across multiple collections. Fusion provides a set of built-in pipelines. You can use the Index Workbench or the *REST API* to develop custom index pipelines to suit any datasource or application.



Fusion creates a default index pipeline when you create an app. The index pipeline has the same name as the app. This pipeline consists of a *Field Mapping index stage*.

Index Pipeline Stages

An Index Pipeline takes content and transforms it into a document suitable for indexing by Solr via a series of modular operations called stages. The objects sent from stage to stage are PipelineDocument objects. Fusion provides many specialized index stages as well as a *JavaScript Index stage* that allows for custom processing via a JavaScript program. The general outline of the Extract/Transform/Load processing performed by an index pipeline is:

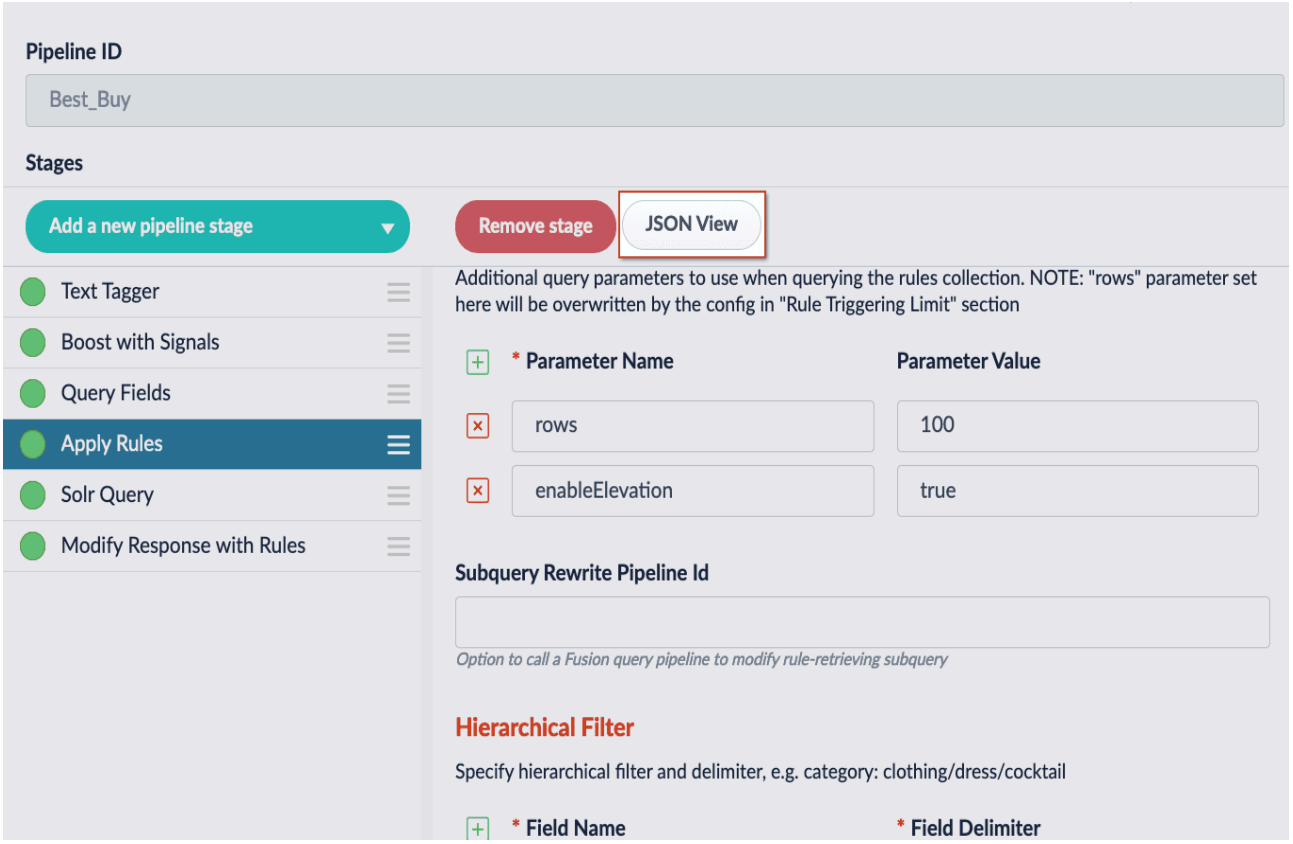
- Raw content is parsed into one or more PipelineDocument objects.
- Any number of intermediate stages operate on the document fields directly, or, in the case of specialized NLP tools, add annotations to a document.
- Finally, the PipelineDocument is sent to Solr for indexing.

A pipeline stage definition associates a unique ID with a set of properties. Pipeline definitions are stored in ZooKeeper for reuse across pipelines and search applications. The Fusion UI provides stage-specific panels used to define and configure each pipeline stage. Alternatively, JSON can be used to specify the sequence of pipeline stages and registered via the Fusion REST API. Some stages require additional resources, e.g., text files that contain lists of names, synonyms, places, or binary files which NLP language models. These resources can be uploaded via the Fusion UI or the REST API.

Pipeline stage JSON editor

The pipeline stage JSON editor gives the ability to create and copy pipeline stages by pasting JSON objects in the Fusion UI. Only JSON is supported, and JSON validation is included to prevent the user from saving an invalid object.

Navigate to your pipelines, select a pipeline stage, and click the **JSON View** button to open the editor:



Existing stages are considered **READ-ONLY**. They will have only a copy button which can be used to create stages. New stages will have the option to copy or paste valid JSON. Changes are applied to the stage with the **Apply** button.

Users will still need to **Save** the stage for the changes made in the editor to be saved.

The editor has several elements worth noting:

Elements	Description
	Copies the JSON to the clipboard.
	Pastes the clipboard into the editor. Only valid JSON is accepted.
	Expands the editor to a fullscreen view.
	Condenses the editor to a compact view.
	Applies the JSON in the editor and updates the stage. The stage must be saved to preserve the changes.
	Resets changes made since the last time changes were applied.

For instructions, see *Use the Pipeline Stage JSON Editor*.

Index Profiles

Index profiles let your applications *send documents for indexing to a consistent endpoint* (the index profile endpoint) and change the underlying index pipeline as needed.

An index profile is also a simple way to use one pipeline for multiple collections, without any one collection "owning" the pipeline. Associating a profile with an index pipeline, or an index pipeline and a collection, is simply a mapping.

Finally, you can also specify configuration parameters for an index pipeline in the index profile.

You can use the API or UI to manage index profiles.

Index Profiles in the API

- *Index Profiles API* (`/index-profiles`)
  Create, read, update, and delete index profiles.

Index Profiles in the UI

To configure index profiles, click **Indexing** > **Index Profiles**.

Custom JavaScript Stages for Index Pipelines

The JavaScript Index stage allows you to write a custom processing logic using JavaScript to manipulate Pipeline Documents and the index pipeline context, which will be compiled by the JDK into Java bytecode that is executed by the Fusion pipeline. The first time that the pipeline is run, Fusion compiles the JavaScript program into Java bytecode using the JDK's JavaScript engine.

> Users who can create or modify code obtain access to the broader Fusion environment. This access can be used to create intentional or unintentional damage to Fusion.

For a JavaScript Index stage, the JavaScript code must return either:

- A single document or array of documents
  or
- The null value or an empty array.
  In the latter case, no further processing is possible, which means that the document will not be indexed or updated. For example, Solr commits have a `null` value and are dropped. For information about how to skip stages when Solr commits are sent, see the *Skip JavaScript stages during Solr commits*.

JavaScript Index Stage Global Variables

JavaScript *(http://en.wikipedia.org/wiki/JavaScript)* is a lightweight scripting language. The JavaScript in a JavaScript stage is standard ECMAScript *(http://en.wikipedia.org/wiki/ECMAScript)*. What a JavaScript program can do depends on the container in which it runs. For a JavaScript Index stage, the container is a Fusion index pipeline. The following global pipeline variables are available:

Name	Type	Description
`doc`	*Pipeline Document*	The contents of each document submitted to the pipeline.
`ctx`	*Context*	A map that stores miscellaneous data created by each stage of the pipeline. > Use the `ctx` variable instead of the deprecated `_context` global variable. The `ctx` variable is used to: • Pass data from one stage to another • Store data that needs to be passed from one custom stage to a later custom stage The data can differ between stages: • If the previous stage changes the data • Based on the configuration of each stage > If the data is modified in one stage, it may cause a later stage to function irregularly.
`collection`	String	The name of the Fusion collection being indexed or queried.
`solrServer`	*BufferingsolrServer*	The Solr server instance that manages the pipeline's default Fusion collection. All indexing and query requests are done by calls to methods on this object. See solrClient *(https://lucene.apache.org/solr/5_2_1/solr-solrj/org/apache/solr/client/solrj/SolrClient.html)* for details
`solrServerFactory`	*solrClusterComponent*	The SolrCluster server used for lookups by collection name which returns a Solr server instance for that collection. For example: `var productsSolr = solrServerFactory.getSolrServer("products");`.

Syntax Variants

JavaScript stages can be written using legacy syntax or function syntax. The key difference between these syntax variants is how the "global variables" are used. While using legacy syntax, these variables are used as global variables. With function syntax, however, these variables are passed as function parameters.

Legacy Syntax

Legacy syntax is used to perform very simple tasks.

```
//do some work... return doc;
```

Function Syntax

Function syntax is used for moderately complex tasks.

```
function (doc) { // do some work ... return doc; }
```

Function syntax is used for the examples in this document.

Advanced Syntax

Advanced syntax is used for complex tasks and when multiple functions are needed.

```
/* globals Java, logger*/ (function () { "use strict"; return function main(doc ,
ctx, collection, solrServer, solrServerFactory) { // do some work ... return doc;
}; })();
```

JavaScript Use

The JavaScript in a JavaScript Index stage must return either a single document or an array of documents. This can be accomplished by either:

- a series of statements where the final statement evaluates to a document or array of documents
- a function that returns a document or an array of documents

All pipeline variables referenced in the body of the JavaScript function are passed in as arguments to the function. For example, in order to access the PipelineDocument in global variable 'doc', the JavaScript function is written as follows:

```
function (doc) { // do some work ... return doc; }
```

The allowed set of function declarations are:

```
function doWork(doc) { ... return doc; } function doWork(doc, ctx) { ... return
doc; } function doWork(doc, ctx, collection) { ... return doc; } function
doWork(doc, ctx, collection, solrServer) { ... return doc; } function doWork(doc,
ctx, collection, solrServer, SolrClusterComponent) { ... return doc; }
```

The order of these arguments is according to the (estimated) frequency of use. The assumption is that most processing only requires access to the document object itself, and the next-most frequent type of processing requires only the document and read-only access of some context parameters. If you need to reference the solrServerFactory global variable, you must use the 5-arg function declaration.

In order to use other functions in your JavaScript program, you can define and use them, as long as the final statement in the program returns a document or documents.

Global variable `logger`

The logs are output to the indexing service logs for custom index stages. Access the Log Viewer and filter on this service to view the information.

The JavaScript Engine Used by Fusion

The JavaScript engine used by Fusion is the Nashorn engine from Oracle. See The Nashorn Java API *(https://docs.oracle.com/javase/8/docs/technotes/guides/scripting/nashorn/api.html)* for details.

Creating and accessing Java types

The following information is taken from Oracle's JavaScript programming guide section 3, Using Java From Scripts *(https://docs.oracle.com/javase/8/docs/technotes/guides/scripting/prog_guide/javascript.html#A1147207)*.

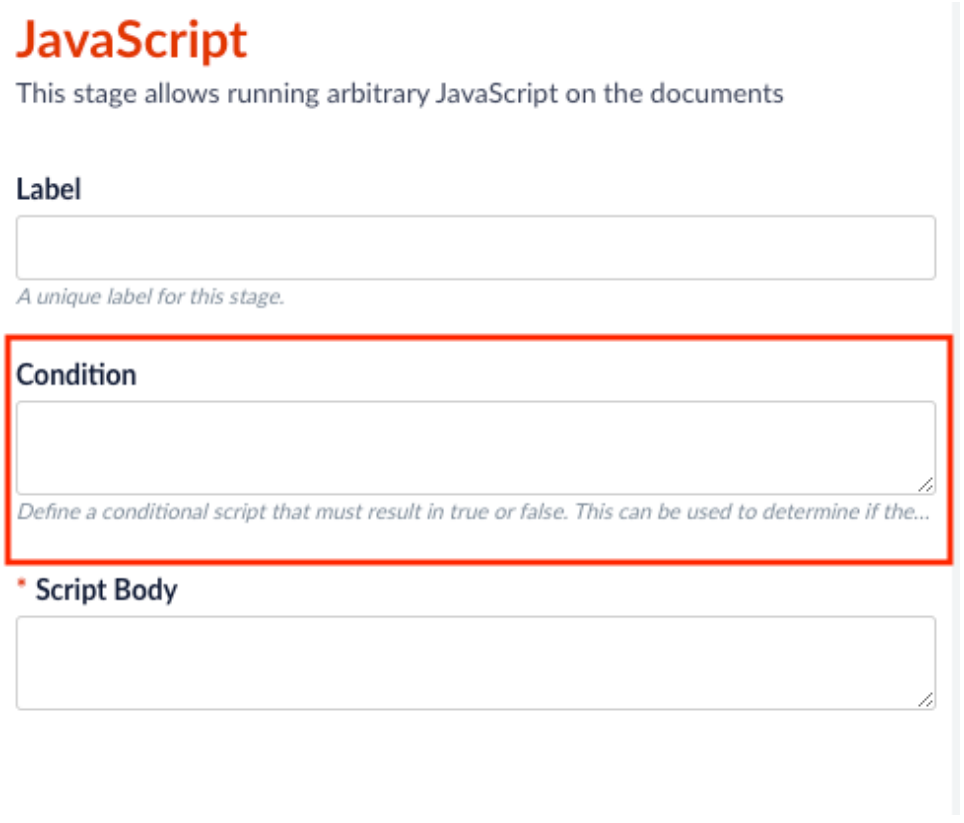
To create script objects that access and reference Java types from Javascript use the `Java.type()` function:

```
var ArrayList = Java.type("java.util.ArrayList"); var a = new ArrayList;
```

Examples

Set the condition field

The JavaScript Index stage lets you define a condition to trigger the script body.



The condition field is evaluated as either true or false. Do not precede with `if` or include `;` at the end of the line.

- Works:
  - `doc.hasField("title_s") === true`
  - `doc.hasField("title_s") === false`
  - `doc.hasField("title_s")`
- Does not work:

- `if doc.hasField("title_s") === false;`

Skip JavaScript stages during Solr commits

When JavaScript stages contain data manipulation such as values in the `conditions` field, errors may be generated and Solr commits are dropped because the value is `null`. To skip those stages when a Solr commit is sent through the pipeline, add the following condition to those stages: `doc.getCommands().size()==0`.

Add a field to a document

```
function (doc) { doc.addField('some-new-field', 'some-value'); return doc; }
```

Join two fields

The following example conjoins separate latitude and longitude fields into a single geo-coordinate field, whose field name follows Solr schema conventions and ends in "_p". It also removes the original latitude and longitude fields from the document.

```
function (doc) { var value = ""; if (doc.hasField("myGeo_Lat") &&
doc.hasField("myGeo_Long")) { value = doc.getFirstFieldValue("myGeo_Lat") + "," +
doc.getFirstFieldValue("myGeo_Long"); doc.addField("myGeo_p", value);
doc.removeFields("myGeo_Lat"); doc.removeFields("myGeo_Long");
logger.debug("conjoined Lat, Long: " + value); } return doc; }
```

Return an array of documents

```
function (doc) { var subjects = doc.getFieldValues("subjects"); var id =
doc.getId(); var newDocs = []; for (i = 0; i < subjects.size(); i++) { var pd = new
com.lucidworks.apollo.common.pipeline.PipelineDocument(id+'-'+i );
pd.addField('subject', subjects.get(i)); newDocs.push( pd ); } return newDocs; }
```

Parse a JSON-escaped string into a JSON object

While it is simpler to use a JSON Parsing index stage, the following code example shows you how to parse a JSON-escaped string representation into a JSON object.

This code parses a JSON object into an array of attributes, and then find the attribute "tags" which has as its value a list of strings. Each item in the list is added to a multi-valued document field named "tag_ss".

```
function (doc, ctx) { /** * Returned body is a JSON string. Example: { "telescope":
"Hubble", "location": "low earth orbit", "primary_mirror_size": "2.4m" "tags":
["launched 1990","repaired 1993","repaired 1997","repaired 1997","repaired
1999","repaired 2002","repaired 2009"] } */ for (var key in myData) { var obj =
myData[key]; if (obj == "tags") { logger.debug("extracting tags from tag field {}",
JSON.stringify(obj)) for (var i = 0; i < obj.length; i++) { var tag = obj[i][0];
doc.addField("tag_ss", tag); } } } return doc; }
```

Do a lookup on another Fusion collection

```
function doWork(doc, ctx, collection, solrServer, solrServerFactory) { var sku = doc.getFirstFieldValue("sku"); if (!doc.hasField("mentions")) { var mentions = "" var productsSolr = solrServerFactory.getSolrServer("products"); if( productsSolr != null ){ var q = "sku:"+sku; var query = new org.apache.solr.client.solrj.SolrQuery(); query.setRows(100); query.setQuery(q); var res = productsSolr.query(query); mentions = res.getResults().size(); doc.addField("mentions",mentions); } } return doc; }
```

Reject a document

If the function returns `null` or an empty array, it will not be indexed or updated into Fusion.

```
function doWork(doc) { if (!doc.hasField("required_field")) { return null; } return doc; }
```

Drop a document by ID

```
function(doc) { var id = doc.getId(); if (id !== null) { var pattern = "https://www.mydomain.com/links/contact/?"; // 0 means the pattern was found so drop the doc return (id.indexOf(pattern) == 0) ? null : doc; } return doc; }
```

Format Date to Solr Date

```
// For example: // From: 26/Mar/2015:14:38:48 -0700 // To: 2015-03-26T14:38:48Z (Solr format) function(doc) { if (doc.getId() !== null) { var inboundPattern = "dd/MMM/yyyy':'HH:mm:ss Z"; // modify this to match the format of the inbound date var solrDatePattern = "yyyy-MM-dd'T'HH:mm:ss'Z'"; // leave this alone var dateFieldName = "apachelogtime"; // change this to your date field name var solrFormatter = new java.text.SimpleDateFormat(solrDatePattern); var apacheParser = new java.text.SimpleDateFormat(inboundPattern); var dateString = doc.getFirstFieldValue(dateFieldName); logger.info("**** dateString: " + dateString); var inboundDate = apacheParser.parse(dateString); logger.info("**** inboundDate: " + inboundDate.toString()); var solrDate = solrFormatter.format(inboundDate); logger.info("**** solrDate: " + solrDate.toString()); doc.setField(dateFieldName, solrDate.toString()); } return doc; }
```

Replace whitespace and newlines

```
function(doc) { if (doc.getId() !== null) { var fields = ["col1", "col2", "col3"]; for (i = 0; i < fields.length; i++ ) { var field = fields[i]; var value = doc.getFirstFieldValue(field); logger.info("BEFORE: Field " + field + ": *" + value + "*"); if (value != null) { value = value.replace(/^\s+/, ""); // remove leading whitespace logger.info("AFTER: Field " + field + ": *" + value + "*"); value = value.replace(/\s+$/, ""); // remove trailing whitespace logger.info("AFTER: Field " + field + ": *" + value + "*"); value = value.replace(/\s+/g, " "); // multiple whitespace to one space logger.info("AFTER: Field " + field + ": *" + value + "*"); doc.setField(field, value); } } } return doc; }
```

Split the values in a field

```
//Split On a delimiter. In this case, a newline function(doc){ if (doc.getId() !== null) { var fromField = "company2_ss"; var toField = "company2_ss"; var delimiter = "\n"; var oldList = doc.getFieldValues(fromField); var values = []; // parse the entries one at a time doc.removeFields(toField); // clear out the target field for (i = 0; i < oldList.size(); i++) { values[i] = oldList.get(i); // get the list of strings split by the delimiter newList = values[i].split(delimiter); for(j = 0; j < newList.length; j++ ){ doc.addField(toField, newList[j]); } } } return doc; }
```

Prevent global variables in JavaScript

Variable declared using the `var` keyword

If a variable is declared using the `var` keyword, the JavaScript interpreter processes the value sequentially.

In this example, the values for `var i = 0` are logged in order as `0`, `1`, `2`, `3`, `4`, etc.

```
var queries = ["cat","the cat", "the cat in the hat","the cat in the hat is back"] for(var i = 0; i < queries.length; i++){ logger.info("query {} is '{}'",i,queries[i]) }
```

Variable not declared using the `var` keyword

If a variable is not declared using the `var` keyword, the JavaScript interpreter moves the declaration of variable and functions to the top of the declared (global) scope. Because Fusion pipeline stages execute in a multi-threaded environment, these global (shared) variables make the stages not thread-safe.

For more detailed information, see Hoisting *(https://developer.mozilla.org/en-US/docs/Glossary/Hoisting#:~:text=JavaScript%20Hoisting%20refers%20to%20the,defined%20in%20the%20ECMAScript%20specification)*.

```
var queries = ["cat","the cat", "the cat in the hat","the cat in the hat is back"] for(i = 0; i < queries.length; i++){ logger.info("query {} is '{}'",i,queries[i]) }
```

Issues and errors that may occur include:

For multi-threaded environments, the value of `i` may not proceed sequentially from 0 to 4 as the loop is processed. Instead, values may be logged based on the execution state of the other pipeline requests. For example, `0`, `1`, `3`, `1`, `2`, etc., which logs the values as `"cat", "the cat", "the cat in the hat is back", "the cat", "the cat in the hat"`.

However, if only one thread is incrementing the `i` variable, the values proceed sequentially (`0`, `1`, `2`, `3`, `4`, etc.)

If the `queries` array varies in length from document to document, the loop may generate an `ArrayIndexOutOfBounds` exception for a Java array or an `undefined` error for a JavaScript array.

Threads may not log all four queries.

Setting the `"use strict"` directive

Setting the `"use strict"` directive tells the JavaScript engine to require non-global declarations of all functions and variables.

The following example demonstrates how to create a copy of a `PipelineDocument` and return both the original and the copy to the pipeline for processing.

```
/* globals Java, logger*/ // This is an optional line that informs the syntax checker about known global objects. This avoids excessive flagging of what it perceives as errors. (function () { //Defines a top-level scope "use strict"; // This directs the interpreter to require `var` for all variable declarations to help ensure thread safety. Any global scope variables generate compile errors. /*** * NOTE: All code within the outer scope, (everything from other than the `main` function returned by the `function` declared on line two) is only run when the JavaScript engine JIT-compiles the script. Place comments, static declarations, functions, and initialization code here to improve readability and performance. */ var PipelineDocument = Java.type('com.lucidworks.apollo.common.pipeline.PipelineDocument'); var ArrayList = Java.type("java.util.ArrayList") /** * Take a PipelineDocument like the one passed to this stage and clone it. Then give it a new id. */ function clonePipelineDoc(pipelineDoc, id) { var clone = new PipelineDocument(pipelineDoc); if (id) { clone.setId(id); } logIfDebug("Cloned document with id '{}' and gave the clone id '{}'",pipelineDoc.getId(),id) return clone; }; var isDebug = false // This setting is optional and determines if log messages are turned off or on. It works in conjunction with the `logIfDebug` function. function logIfDebug(m){if(isDebug && m)logger.info(m, Array.prototype.slice.call(arguments).slice(1));} return function main(doc , ctx, collection, solrServer, solrServerFactory) { //This returns the function to be called by Fusion when the stage executes, and guarantees Fusion will always call this function. When a stage does not use advanced syntax, but declares multiple top-level functions, Fusion may not be able to determine which function to call. Duplicate doc and give it an ID that is the reverse of the original. var doc2 = clonePipelineDoc(doc, doc.getId().split("").reverse().join("")) var list = new ArrayList() list.add(doc) list.add(doc2) //Return both docs to the downstream stages. return listOfDocs; }; })();
```

Additional resources

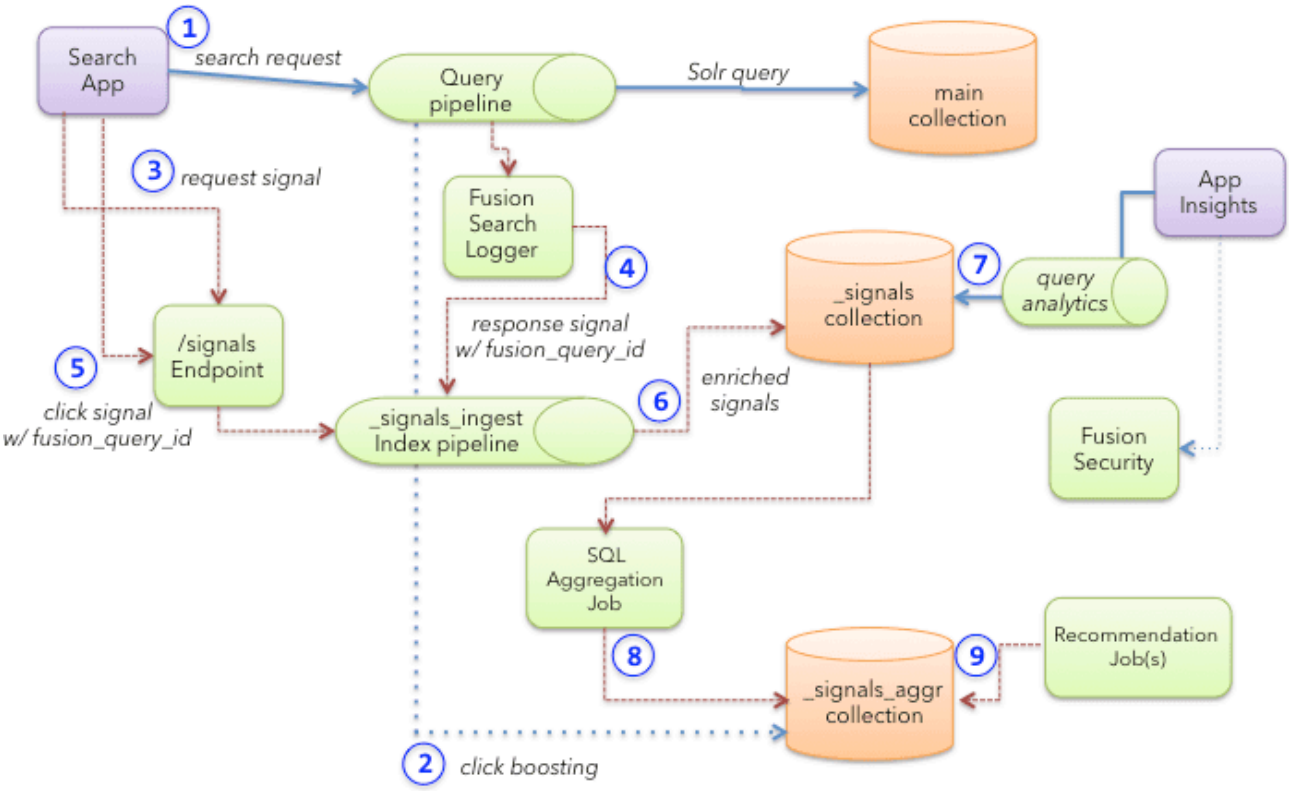
- View our training on JavaScript in Fusion *(https://academy.lucidworks.com/javascript-in-fusion)*.
- *Fusion Pipeline Javadocs*

Indexing Signals

Signals are indexed just like other data, but instead of using a connector, you use the *Signals API*.

Signals data flow

This diagram shows the flow of signals data from the search app through Fusion. The numbered steps are explained below.



1. The search app sends a query to a Fusion query pipeline.
   The query request should include a user ID and session query parameter to identify the user.
2. Optionally, the Fusion query pipeline queries the `COLLECTION_NAME_signals_aggr` collection to get boosts for the main query based on aggregated click data.
3. The search app also sends a request signal to the Fusion `/signals` *endpoint*.
   The primary intent of a request signal is to capture the raw user query and contextual information about the user's current activity in the app, such as the user agent and the page where they generated the query. The request signal does not contain any information about the results sent to Solr; it is created before a query is processed.
4. Once Solr returns the response to Fusion, the SearchLogger component indexes the complete request/response data into the `COLLECTION_NAME_signals` collection as a response signal using the `_signals_ingest` pipeline. Therefore, the response signal captures all results from Fusion as it related to the original query.

   > This is a departure from pre-4.0 versions of Fusion where query impressions were logged in a separate `COLLECTION_NAME_logs` collection. Query activity is no longer indexed into the `_logs` collection. All response signals use the `fusion_query_id` (see below) as the unique document ID in Solr.

5. When the user clicks a link in the search results, the search app sends a click event to the Fusion signals endpoint (which invokes the `_signals_ingest` pipeline behind the scenes).
   The click signal *must* include a field named `fusion_query_id` in the `params` object of the raw click signal. The `fusion_query_id` field is returned in the query response (from step 1) in a response header named `x-fusion-query-id`. This allows Fusion to associate a click signal with the response signal generated in step 4. The `fusion_query_id` is also used by Fusion to associate click signals with *experiments*. For experiments to work, each click signal must contain the corresponding `fusion_query_id` that produced the document/item that was clicked.
6. The `_signals_ingest` pipeline enriches signals before indexing into the `COLLECTION_NAME_signals` collection.
   This enrichment includes field mapping, geolocation resolution, and updating the `has_clicks` flag to "true" on request signals when the first click signal is encountered for a given request using the *Update Related Document index stage*.
7. Fusion queries the `COLLECTION_NAME_signals` collection through a Fusion query pipeline to generate query analytics reports from raw signals.
8. Behind the scenes, the *SQL aggregation* framework aggregates click signals to compute a weight for each query + `doc_id` + filters group.

The resulting metrics are saved to the `COLLECTION_NAME_signals_aggr` collection to generate boosts on queries to the main collection (step 2 above).

9. Recommendations also use aggregated documents in the `COLLECTION_NAME_signals_aggr` collection to build a *collaborative filtering-based recommender model*.

Default index pipeline for signals

When indexing signals of any type for any Fusion app, Fusion always uses a default index pipeline named `_signals_ingest` unless you explicitly specify a different index pipeline.

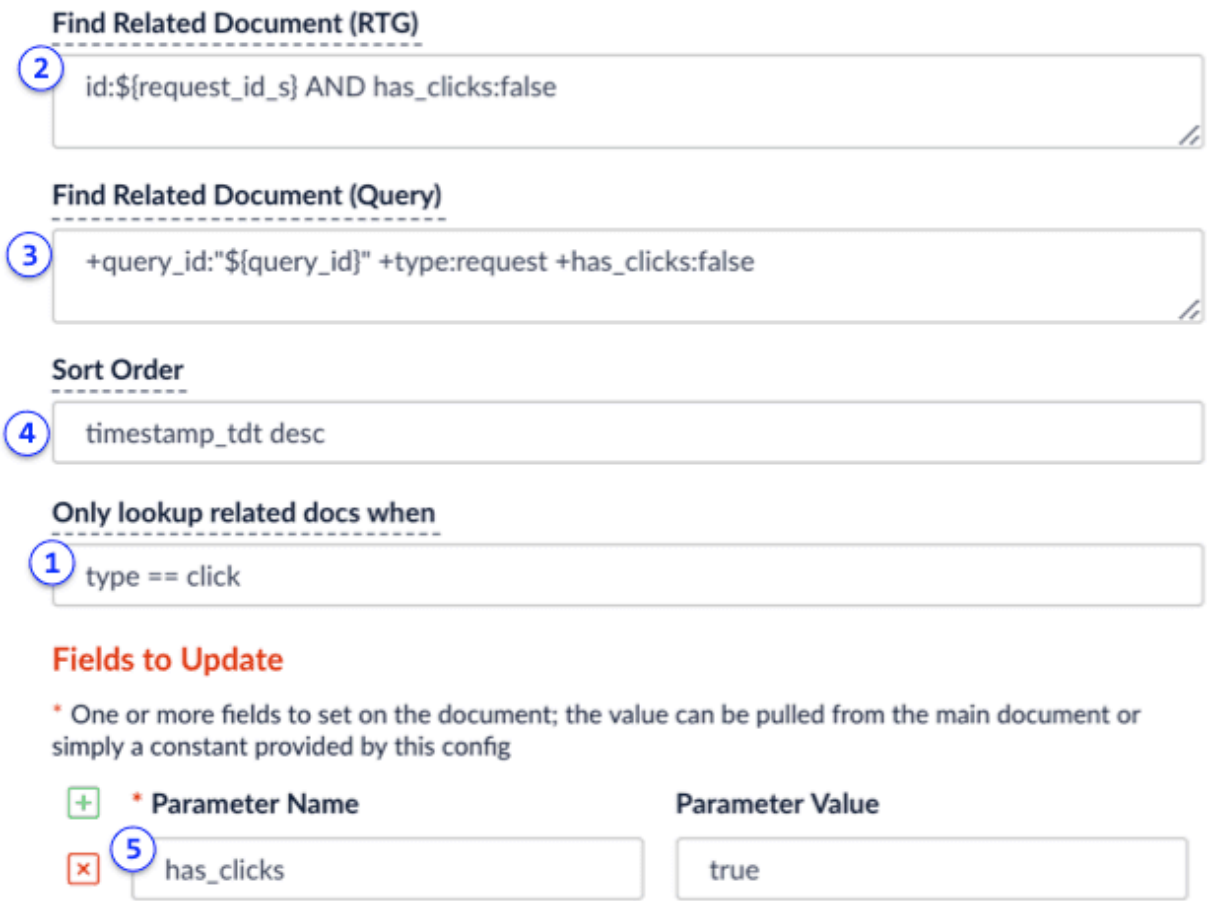
	Because this pipeline is not associated with any Fusion app, it does not automatically appear in the list of index pipelines. You can find it in the *Object Explorer* by clicking the **In No Apps** filter.

Default stages

The `_signals_ingest` index pipeline has several stages:

1. *Format Signals stage*
2. *Field Mapping stage*
3. *GeoIP Lookup stage*
4. *Solr Indexer stage*
5. Update `has_clicks` flag stage
   The Update `has_clicks` flag stage is an instance of the *Update Related Document stage* that updates the `has_clicks` flag to "true" on an existing request signal after the first click signal is processed for the request.

The Update `has_clicks` flag stage works as follows:



1. When a click signal is encountered (`type==click`)
2. Look at the incoming click signal for a field named `request_id_s`, which gets set by the Format Signals stage using a distributed cache of recently processed request signals.
   If the `request_id_s` field is set, then send a real-time `GET` query to Solr to find a request signal with ID equal to the value of the `request_id_s` field on the click signal. To avoid re-updating request signals, the RTG query also filters on `has_clicks==false`, which avoids duplicate atomic updates on the same document in Solr. Real-time `GET` is used to avoid timing issues between a request signal being sent to Solr and when it gets committed. This prevents missing updates when clicks occur soon after the initial request signal is sent by the search app.

3. If the click signal does not have the `request_id_s` field set, then do a normal Solr lookup for the request signal using: `+query_id:"${query_id}" +type:request +has_clicks:false`. A click signal may not have a `request_id_s` if there is a cache miss in the distributed cache used by the Format Signals stage.
4. If the stage performs a normal query, there may be multiple request signals that have the same `query_id`. This is because the `query_id` is based on `session` + `query` + `filter`, so if a user sends the same `query` + `filter` during the same session, there will be multiple request signals with the same `query_id` value. Thus, the stage sorts to get the latest request signal to update.
5. If a related document is found (in this case a request signal), then the stage updates the `has_clicks` field to true and performs an atomic update in Solr.

This stage performs its work in a background thread, so it does not impact the indexing performance of the click signal.

Data Models

Data models simplify the process of getting started with Fusion by providing pre-configured objects to reduce the effort spent on basic starting tasks. This helps keep documents consistent between datasources and intuitive to the object's type.

Components

Data models consist of the following:

- ***Object type*** - Describes the object type, or what a document in the index is. For example, an object may be a file, email, person, product, etc.
- ***Index pipeline*** - Specifies the index pipeline that is indexing documents relevant to the data model.
- ***Query pipeline*** - Specifies the query pipeline that is invoked when a query involves the data model.
- ***Fields configuration*** - Creates a schema map that describes how the object schema should be translated into the Solr index. For example, the field name `first_name` can map to Solr fields `first_name_s` and `first_name_t`.

Object type

The object type defines what a document is and what associated data is expected.

Example
When indexing employees for a company directory, certain fields are expected by a user. This includes the employees's name, title, and contact information. An object type, `person`, is created for this purpose. The data model object is created to include this information in a clear, intuitive way:

Data Model Field	Solr Fields
`first_name`	`firstName_t` `firstName_s`
`last_name`	`lastName_t` `lastName_s`
`email`	`email_s`
`job_title`	`jobTitle_t` `lastName_s`

Index pipeline

When you add a connector in Fusion, a default index pipeline is created that is used to map the documents' fields to the data model fields. The pipeline contains two pipeline stages, the ***Data Model Mapping*** *index stage* and the ***Call Data Model Pipeline*** *index stage*. Once configured, this stage indexes the documents as various object types. Fields irrelevant to that object type are optionally discarded.

Query pipeline

The query pipeline is applied by default whenever a user is searching documents of the object type. This enables query pipeline configurations specialized for the object type. For example, a query pipeline for the object type `person` could boost or filter results for current employees over previous employees.

	The query pipeline is optional for data models.

Fields configuration

Data model fields allow you to map document fields to Solr fields in a way that respects the data model object type. The fields configuration allows this map to be intuitive to new users, offering a guided experience for configuring these options.

UI fields configuration

Field	Required	Description	Example
Name of the Data Model	✅	`string` A user-friendly name for the data model. This typically describes the object type.	`person`
Index Pipeline	✅	`string` The name of the index pipeline used for the data model.	`companyDirectory-index-pipeline`
Query Pipeline	✅	`string` The name of the query pipeline used for the data model.	`companyDirectory-query-pipeline`
Field Name	✅	`string`	`first_name`
Required	✘	`boolean`	`true`
Solr Fields ↓			
Solr Field Name	✅	`string`	`firstName_t`
Is Query Field	✘	`boolean`	`true`
Enable Phrase Match	✘	`boolean`	`false`
Boost Value	✘	`integer` The amount of boost to give to the query.	`1`
Phrase Boost	✘	`integer` The amount of boost to give to the query, if it matches as an exact phrase.	`2`

More information

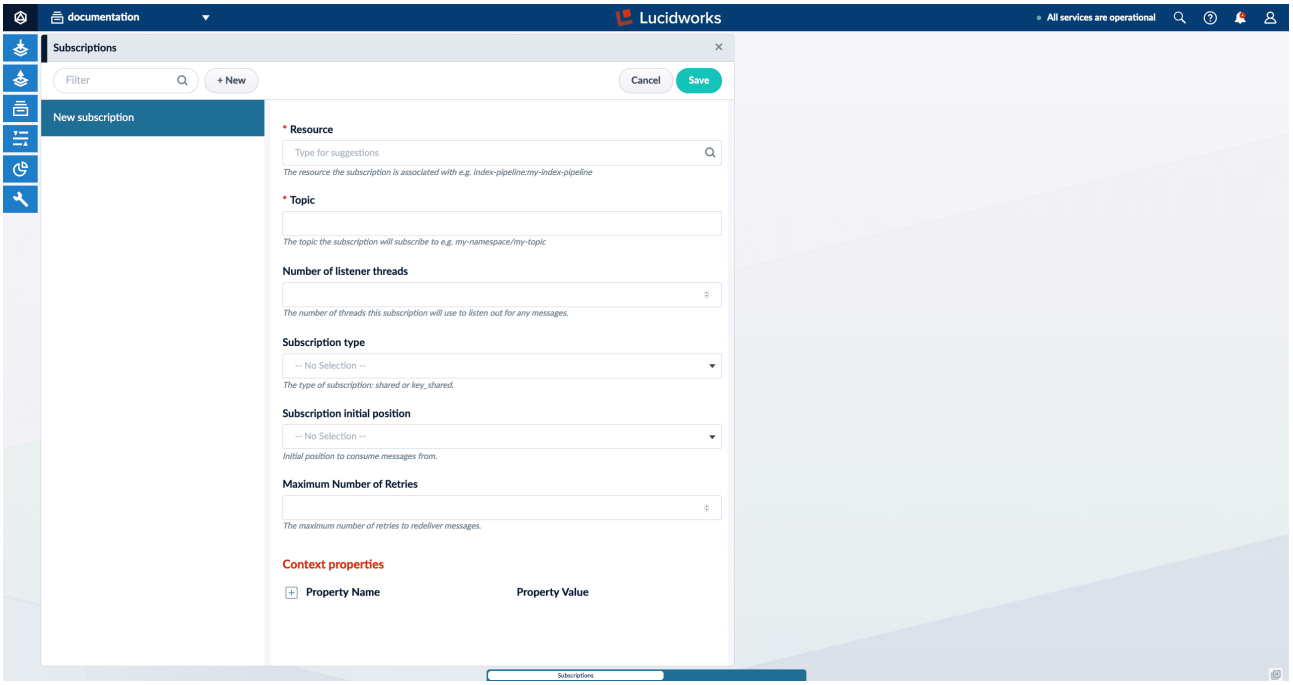
- *How to Configure a Data Model*
- *Data Models API reference*
- *Data Model Mapping Index Stage reference*
- *Call Data Model Pipeline Index Stage reference*

Subscriptions UI

Subscriptions in Fusion allow you to create and configure subscriptions using *Apache Pulsar*.

With the release of Fusion 5.4, an existing subscription's Subscription Type can no longer be changed via the UI or API. Delete existing subscription and create new subscription when subscription type change is desired.

To use Subscriptions in the Fusion UI, navigate to **Indexing > Subscriptions** and click the **New** button.



More information

- *Apache Pulsar in Fusion*
- *Configurations and API reference information*.

Unresolved directive in <stdin> - include::/fusion/dev-portal/api/misc/subscriptions-api.asciidoc[tag=ui-configs]

Apache Pulsar

Basics

- Pulsar gives you the ability to subscribe a pipeline to a topic
- Topics belong to namespaces
  - At the namespace level, you can specify retention policies and schema enforcement
- Namespaces belong to tenants
  - Tenants can specify authorization/authentication

Highlights

- Supports native streaming transformations through Pulsar Functions
- Uses Apache ZooKeeper for configuration management
- Uses Apache BookKeeper for durable storage
- Each document processed has a result object persisted to another topic

Overview

Apache Pulsar is a message bus that combines queuing with publishing-subscribing features for server-to-server messaging. A producer publishes a message to a topic and a consumer subscribes to that *topic* using a *subscription type* to receive the topic messages.

	With the release of Fusion 5.4, an existing subscription's Subscription Type can no longer be changed via the UI or API. Delete the existing subscription and create a new subscription when you want to change the subscription type.

Thanks to Pulsar's messaging guarantees (and Apache ZooKeeper), messages that have not been acknowledged by a consumer will be persisted and re-published. This means saves to crawl-db collection can be done once when an item is completed (versus stored in Solr first, queried for, and then fetched).

Apache Pulsar is a **multi-tenant messaging system**. Several tenants can use the message bus concurrently. Each tenant can have multiple namespaces and each namespace can have multiple topics.

	Even though Pulsar itself is capable of moving data between clusters, Fusion does not support configuring Apache Pulsar to perform geo-replication.

Types of server-to-plugin calls:

- **Broadcast (pub/sub).** The server publishes a message to a topic. Each plugin instance consumes the message through its own unique, *exclusive* subscription. In this scenario, all plugin instances for a given `configId` consume the same message. Examples include job actions (start, stop) and health status requests.
- **Queueing.** The server publishes messages to a topic, where plugins consume through a *shared* subscription. Subscriptions that are *shared* allow multiple consumers to subscribe and consume messages one by one. Shared subscriptions are needed for distributed fetching.
- **Failover.** The connector job service processes messages from a plugin output topic. This is just like *exclusive* (queueing), but allows another server instance to take over if the current instance fails.

Fusion Integration

Fusion will create a Pulsar namespace for every app that is created. The default tenant name for Pulsar is `{kubernetes-namespace}`. Pulsar does not store messages if there are no active subscriptions for the topic. Due to this, namespaces are configured for message retention with a default time of 1 day and a default size of 5GB.

Messages for topics that have subscriptions but no active consumers will be stored in the backlog. Backlog quota is managed per topic.

Pulsar is used by default for Signals. `_signals_ingest` **pipeline** has a subscription to the `_signals_ingest` **topic** in the `_system` **namespace**. Signals belong to the `_system` app that is created in Fusion when the API service boots up.

	Subscriptions are integrated with the Fusion UI. See *Subscriptions UI* for more information.

Troubleshooting Apache Pulsar in Fusion

See *Troubleshooting Apache Pulsar Issues* for more information.

Frequently asked questions

What is the difference between backlogs and retention policy?

Backlogs

Backlogs are:

- Sets of unacknowledged messages for a topic.
- Stored by Pulsar bookies until the messages are processed and acknowledged.
- Governed by the following policies:
  - Backlog quota. An allowable size threshold for each topic in the namespace.
  - Backlog retention. The action the broker takes if the threshold is exceeded. The actions are `producer_request_hold`, `producer_exception`, and `consumer_backlog_eviction`.

Topic retention policy

Topic retention policy sets the size limit and the amount of time messages are kept after they are acknowledged. These settings are configured on the namespace level and applied per each topic.

What are the backlog and retention storage requirements?

The bookkeeper ensemble requires enough disk space to store messages retained in the backlog and kept after acknowledgement. The backlog and retention policies are applied independently to each topic. To calculate enough space, include the total for both backlog and retention needs across all topics.

By default, the following retention policy values are applied *per each topic*:

- Backlog quota size: 10GB
- Backlog retention policy: `producer_request_hold`
- Topic retention policy size: 0
- Topic retention policy time: 0

By default, Fusion creates the following topics:

- `_system/_signals_ingest`
- `_system/_signals_ingest_res`
- `_system/_fusion_crud_events`
- `_system/_fusion_link_events`
- `_logs/system_logs` with:
  - Backlog quota size: 1GB
  - Backlog retention policy: `consumer_backlog_eviction` is turned *on*

How do I change heap memory settings?

Change the default heap memory settings in the following heap chart values:

- `broker.configData.PULSAR_MEM`
- `bookkeeper.configData.PULSAR_MEM`

What is the difference between broker and bookkeeper, and how do I scale them?

Broker

Broker is a stateless component responsible for:

- Managing Pulsar topics.
- Handling produce and consume operations coming from clients.

Additional information:

- A single Pulsar topic can be assigned to only one broker instance. That instance is responsible for handling client requests for that topic.
- The Pulsar topic-broker assignments must be distributed evenly (load balanced) to ensure optimal performance.
- Broker instances are easily scaled up or down based on the number of topics and message volume in the cluster.

Bookkeeper

Bookkeeper is responsible for:

- Persisting message data.
- Storing messages in a consistent, accessible, and fault-tolerant manner.

Parameter information

The following parameters determine how *each topic* is stored in the bookkeeper namespace persistence setting:

- Ensemble size: Size of the pool of bookkeeper instances available for that ledger, to which Pulsar writes. The Fusion default is 2.
- Write quorum: Number of bookkeepers to which Pulsar writes. The value is less than or equal to the Ensemble size. The Fusion default is 2.
- ACK quorum: Number of bookkeepers that must acknowledge the **write** action for the Pulsar broker to send an acknowledgement to the client. The Fusion default is 2.

Bookkeeper and fragment information

- When a new bookkeeper is added to the cluster:
  - New fragments or ledgers that are created write to the new bookkeeper immediately.
  - Manual load balancing may not be necessary. Use standard procedures to load balance the system.
- Each fragment can be stored on a different subset of bookies in the cluster.
- Topics and ledgers are *not* coupled to a set of bookkeeper nodes.

Rack information

A rack can be a logical construct. For example, an Availability Zone (AZ) in a cloud environment.

A rack-aware policy allows:

- Bookkeepers to be marked as belonging to a specific rack.
- The bookkeeper client of the Pulsar broker to select bookies from different racks.

Brokers on individual nodes

In addition to rack awareness, you can configure individual brokers to use a bookkeeper node from a certain group. See Pulsar configuration *(https://pulsar.apache.org/docs/en/2.6.2/reference-configuration/)* for more information.

Additional Pulsar information

See the following topics for more information:

- Pulsar Overview *(https://pulsar.apache.org/docs/en/next/concepts-overview/)*
- Understanding How Apache Pulsar Works *(https://jack-vanlightly.com/blog/2018/10/2/understanding-how-apache-pulsar-works)*

Connectors and Datasources

Connectors are the out-of-the-box components for pulling your data into Fusion. Lucidworks provides a wide variety of connectors, each specialized for a particular data type. When you add a *datasource* to a *collection*, you specify the connector to use for ingesting data.

Connectors are distributed separately from Fusion. For complete information, see *Fusion Connectors*.

Blob Storage

Fusion accepts large binary objects (blobs) for upload, and stores them in Solr. Blob uploads are used to install models, lookup lists, JDBC drivers, and more.

Blob Types

A `resourceType` query parameter can be used to specify the blob type. For example, specify `file` when uploading a text or CSV file, like this:

```
curl -H 'Content-Type: text/csv' -u USERNAME:PASSWORD -X PUT
'https://FUSION_HOST:FUSION_PORT/api/apps/APP_NAME/blobs/quickstart/EXAMPLE.csv?
resourceType=file' --data-binary @/path/EXAMPLE.csv
```

The complete list of valid values for `resourceType` is below:

Type	Description
`banana`	A Banana *dashboard*
`catalog`	An *analytics catalog*
`driver:jdbc`	A *JDBC driver*
`file:js-index`	A JavaScript file for use with a *Managed Javascript index stage*
`file:js-query`	A JavaScript file for use with a *Managed Javascript query stage*
`file`	Any uploaded file, such as from the *Quickstart* or the *Index Workbench*
`model:open-nlp`	An *OpenNLP model*
`unspecified`	A blob of unknown type If no `resourceType` is specified on upload, "other" is assigned by default.
`plugin:index-stage`	An *index stage plugin*
`plugin:query-stage`	A *query stage plugin*

The Blob manager

In addition to the *Blob Store API*, the Fusion UI provides an interface to the blob store in the *Fusion workspace* at **System > Blobs**.

- Click **Add** to upload a new blob.
- Select an uploaded blob to view, replace, or delete it.
- Click **Add** to upload a new blob.
- Select an uploaded blob to view, replace, or delete it.

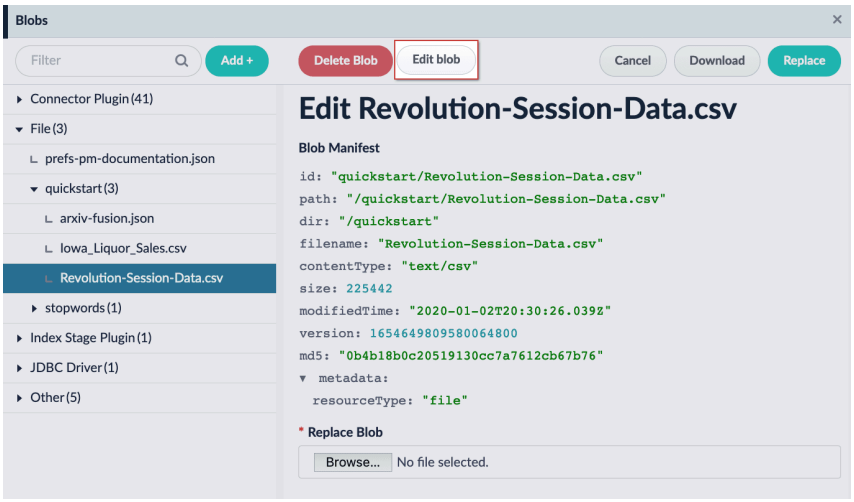
Blob editor

The blob editor gives the ability to edit blobs in the Fusion UI in a variety of formats:

- CSV
- Java

- JavaScript
- Python
- SCALA
- Typescript
- Plain text

Navigate to **System > Blobs**, select a blob, and click the **Edit blob** button to open the editor:



• If the **Edit blob** button is not available, the blob type cannot be edited with the blob editor. • For Fusion 5.4.x and later, JavaScript can be edited directly in the Fusion UI.

The editor has several elements worth noting:

Elements	Description
	Copies the JSON to the clipboard.
	Pastes the clipboard into the editor. Only valid JSON is accepted.
	Expands the editor to a fullscreen view.
	Condenses the editor to a compact view.
	Saves the JSON in the editor and updates the blob.
	Resets changes made since the last time changes were applied.

For instructions, see *Use the Blob Editor*.

Importing Signals

Normally, signals are indexed as streaming data during the natural activity of users. See *Indexing Signals* for details.

For more information, see *Parallel Bulk Loader*.

Parallel Bulk Loader (PBL)

Fusion Parallel Bulk Loader (PBL) jobs enable bulk ingestion of structured and semi-structured data from big data systems, NoSQL databases, and common file formats like Parquet and Avro.

The Parallel Bulk Loader leverages the popularity of Spark as a prominent distributed computing platform for big data. A number of companies invest heavily in building production-ready Spark SQL data source implementations for big data and NoSQL systems, much as Lucidworks has done for Solr. The Parallel Bulk Loader uses connectors provided by the experts who develop these complex systems.

Available data sources

The Parallel Bulk Loader can load documents from any system that implements the Data Sources API for Spark SQL 2.2.1 or later.

These are data sources that the Parallel Bulk Loader can use. For data sources that use Spark SQL connectors, the source of the connector is indicated.

- Solr databases
  Connector (Lucidworks): spark-solr *(https://github.com/lucidworks/spark-solr)*
- Files in these common formats: JSON, CSV, XML, Apache Avro, and Apache Parquet
- JDBC-compliant databases
- Apache HBase databases
  Connector (Hortonworks): Apache Spark - Apache HBase Connector *(https://github.com/hortonworks-spark/shc)*
  Spark-on-HBase *(https://hortonworks.com/blog/spark-hbase-dataframe-based-hbase-connector)*
- Datasets accessible through Apache Hive
- Apache Cassandra NoSQL databases
  Connector (DataStax): Spark-Cassandra connector *(https://github.com/datastax/spark-cassandra-connector)*
- Elastic databases
  Connector: Elasticsearch-Hadoop connector *(https://github.com/elastic/elasticsearch-hadoop)*
  Use the package: `org.elasticsearch:elasticsearch-spark-20_2.11:6.2.2`
- MongoDB databases
- Riak databases
- Couchbase NoSQL databases
  Connector: Couchbase-Spark connector *(https://spark-packages.org/package/couchbase/couchbase-spark-connector)*
  Use the package: `com.couchbase.client:spark-connector_2.11:2.2.0`
- Redis in-memory data structure stores
- Google data sources, including Google Analytics, Sheets, and BigQuery
  Connectors:
  Analytics *(https://github.com/crealytics/spark-google-analytics)*
  Sheets *(https://github.com/potix2/spark-google-spreadsheets)*
  Big Query *(https://github.com/spotify/spark-bigquery)*
- Microsoft Azure DataLake, Cosmos DB, and SQL Database

Key features

Key features of the Parallel Bulk Loader are:

- **Load distribution.** To distribute load and maximize performance, the Parallel Bulk Loader parallelizes operations and distributes them across the Fusion Spark cluster.
- **No parsing.** No parsing is needed. The Spark Data Sources API returns a DataFrame (RDD + schema) that has an easy-to-use tabular format.
- **Dynamic resolution of dependencies.** There is nothing to download or install. Users just provide the Maven coordinates of dependencies during configuration, and Spark distributes the necessary JAR files to worker nodes.
- **Leverage integration libraries.** Similar to JDBC, the Parallel Bulk Loader leverages integration libraries built by the experts of the underlying systems, for example, Databricks, DataStax, Hortonworks, Elastic, Lucidworks, Microsoft, and so forth.
- **Direct write operations.** The Parallel Bulk Loader writes directly to Solr (for maximum performance) or to Fusion index pipelines (for maximum flexibility).
- **Solr atomic updates.** The Parallel Bulk Loader uses atomic updates to update existing documents in Solr.
- **Incremental queries.** To obtain the latest changes to data sources, macros built into the Parallel Bulk Loader use timestamps to filter queries.
- **Seamless integration with Spark-NLP.** Do natural language processing, including part-of-speech tagging, stemming or lemmatization, sentiment analysis, named-entity recognition, and other NLP tasks.
- **SQL joins.** Use SQL to join data from multiple Solr collections.

- **Load Fusion ML models.** To classify incoming documents, load Fusion Machine Learning models stored in the Fusion blob store.
- **SQL transformations.** Leverage the full power of the Spark Scala's DataFrame APIs and SQL to filter and transform data.
- **UDF and UDAF functions.** Select from hundreds of user-defined functions and user-defined aggregate functions.

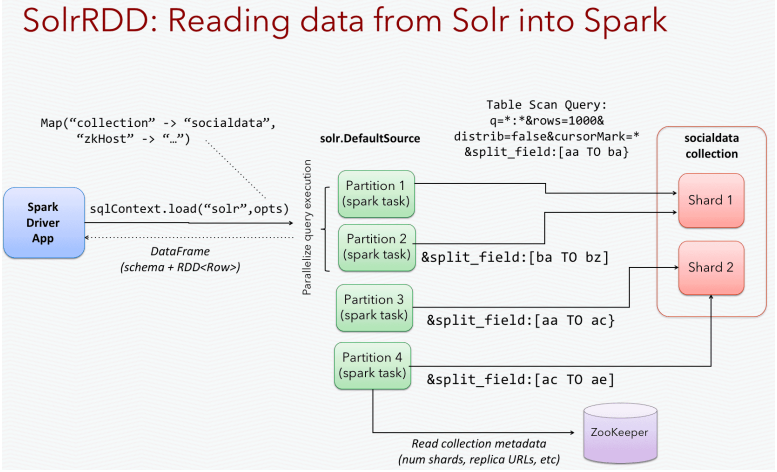
Differences between the Parallel Bulk Loader and Fusion classic connectors

The primary difference between the Bulk Loader and Fusion classic connectors is that the Bulk Loader uses Spark SQL and Spark/Solr integration to perform distributed reads from data sources.

Here are some examples of how the Parallel Bulk Loader performs distributed reads:

- **HBase table.** To support high-volume data ingestion into Solr, the Parallel Bulk loader can distribute queries sent to HBase tables across multiple region servers.
- **Parquet files.** The Parallel Bulk loader processes a directory of Parquet files in HDFS in parallel using the built-in support for computing splits in HDFS files.
- **Spark/Solr integration.** With Spark/Solr integration, the Parallel Bulk Loader uses a Spark-Solr data source *(https://github.com/lucidworks/spark-solr)* to send queries to all replicas of a collection, so it can read from Solr in parallel across the Spark cluster.

This diagram depicts how the Spark-Solr data source partitions queries across all shards/replicas of a Solr collection to better utilize cluster resources and to improve read performance. Most Spark-SQL data sources do something similar for their respective databases, which is one of the main benefits of using the Parallel Bulk Loader job.



In contrast, most classic connectors have only a single-fetcher mode. To scale the fetching with classic connectors, you must distribute the connector itself, which differs from relying on the built-in parallelization of Spark.

Also, most classic connectors rely on Fusion parsers and index pipelines to prepare data for indexing, whereas no parsing is needed for the Parallel Bulk Loader, which can achieve maximum indexing performance by writing directly to Solr.

Getting Data Out

Getting data out of Fusion can mean querying your content or your signals, or accessing other data for analytics.

Querying Fusion

Querying Fusion means sending a request to Fusion's *Query API endpoints*, specifying the query profile and any Solr query parameters. The request is processed by the query pipeline associated with the specified profile, and the response includes any enhancements configured in the query pipeline. You can also enter queries in the *Query Workbench* to test your query pipeline.

Start by learning *the basics of querying*, then about the out-of-the-box *query enhancement features*. For even more sophisticated personalization of search results, see *Advanced Query Enhancement*.

Data analytics

Fusion provides search and analytic functions that allow you to:

- Explore and analyze the data retrieved
- Run standard reports and create custom reports
- Confirm the search results are what you intended to retrieve
- Verify the accuracy of the results
- Enable and run enhanced query options that can power virtual assistants and chatbots
- Run options to assist with predictive analysis

See *Data Analytics* for more information.

Query Basics

Querying

Query Pipelines

Query pipelines process search requests before sending them to Fusion's Solr core. A query pipeline can perform transformations on a search request in order to customize the search results that Solr returns. See *Query Pipeline Configuration*.

Search Applications

Search applications are the front-end interfaces that you build on top of Fusion. Your application makes calls to Fusion's REST API in order to retrieve search results or perform other actions. Certain features, like autocomplete and synonyms, require some configuration on the Fusion back end. See *Application Development*.

Query Language

Whenever you are getting data out of Fusion, you may find it handy to consult the *Query Language Cheat Sheet*.

If you are using Fusion, you can generate and retrieve additional data for analysis or to enhance the end-user experience. For example, Fusion can produce sophisticated recommendations to guide end users to the best available results, including results that do not exactly match the original user-submitted query.

It can also perform machine learning functions that automatically improve search results based on the past activities of users. For more information, see:

- *Natural Language Processing*
- *Machine Learning*
- *Machine Learning Models in Fusion*
- *Machine Learning Jobs*

Fusion SQL

Most organizations that deploy Fusion also have SQL-compliant business intelligence (BI) or dashboarding tools to facilitate self-service analytics.

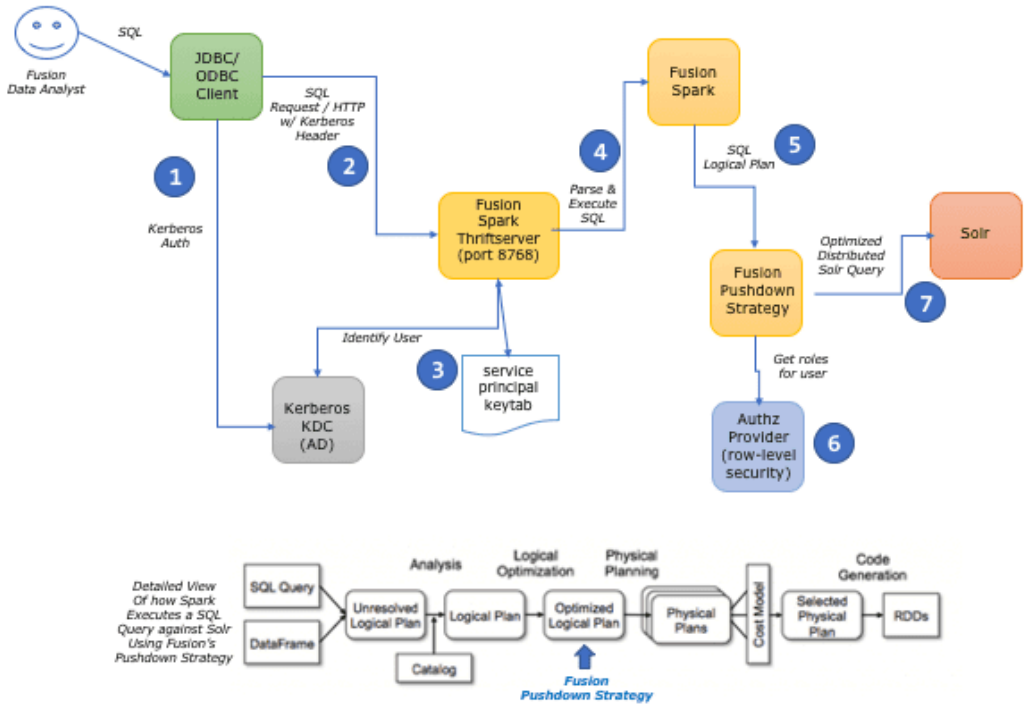
The Fusion SQL service:

- Lets organizations leverage their investments in BI tools by using JDBC and SQL to analyze data managed by Fusion. For example, Tableau is popular data analytics tool that connects to Fusion SQL using JDBC to enable self-service analytics.
- Helps business users access important data sets in Fusion without having to know how to query Solr.

In addition to the specified *System Requirements*, Fusion on Windows requires Visual C++ Redistributable for Visual Studio 2015 *(https://www.microsoft.com/en-us/download/details.aspx?id=48145)* to start the SQL service successfully.

Fusion SQL architecture

The following diagram depicts a common Fusion SQL service deployment scenario using the Kerberos network authentication protocol for single sign-on. Integration with Kerberos is optional. By default, the Fusion SQL service uses Fusion security for authentication and authorization.



The numbered steps in the diagram are:

1. The JDBC/ODBC client application (for example, TIBCO Spotfire or Tableau) uses Kerberos to authenticate a Fusion data analyst.
2. After authentication, the JDBC/ODBC client application sends the user's SQL query to the Fusion SQL Thrift Server over HTTP.
3. The SQL Thrift Server uses the keytab of the Kerberos service principal to validate the incoming user identity.
   The Fusion SQL Thrift Server is a Spark application with a specific number of CPU cores and memory allocated from the pool of Spark resources. You can scale out the number of Spark worker nodes to increase available memory and CPU resources to the Fusion SQL service.
4. The Thrift Server sends the query to Spark to be parsed into a logical plan.
5. During the query planning stage, Spark sends the logical plan to *Fusion's pushdown strategy* component.
6. During pushdown analysis, Fusion calls out to the registered AuthZ FilterProvider implementation to get a filter query to perform row-level filtering for the Kerberos-authenticated user.
   By default, there is no row-level security provider but users can install their own implementation using the Fusion SQL service API.
7. Spark executes a distributed Solr query to return documents that satisfy the SQL query criteria and row-level security filter. To leverage the distributed nature of Spark and Solr, Fusion SQL sends a query to all replicas for each shard in a Solr collection. Consequently, you can scale out SQL query performance by adding more Spark and/or Solr resources to your cluster.

Fusion pushdown strategy

The pushdown strategy analyzes the query plan to determine if there is an optimal Solr query or streaming expression that can push down aggregations into Solr to improve performance and scalability. For example, the following SQL query can be translated into a Solr facet query by the Fusion pushdown strategy:

```
select count(1) as the_count, movie_id from ratings group by movie_id
```

The basic idea behind Fusion's pushdown strategy is it is much faster to let Solr facets perform basic aggregations than it is to export raw documents from Solr and have Spark perform the aggregation. If an optimal pushdown query is not possible, then Spark pulls raw documents from Solr, and then performs any joins or aggregations needed in Spark. Put simply, the Fusion SQL service tries to translate SQL queries into optimized Solr queries. But failing that, the service simply reads all matching documents for a query into Spark, and then performs the SQL execution logic across the Spark cluster.

Which collections are registered

By default, all Fusion collections except system collections are registered in the Fusion SQL service so you can query them without any additional setup. However, empty collections cannot be queried, or even described from SQL, so empty collections will not show up in the Fusion SQL service until they have data. In addition, any fields with a dot in the name are ignored when tables are auto-registered. You can use the Catalog API to alias fields with dots in the names to include these fields.

If you add data to a previously empty collection, then you can execute either of the following SQL commands to ensure that the data gets added as a table:

```
show tables show tables in `default`
```

The Fusion SQL service checks previously empty collections every minute and automatically registers recently populated collections as a table.

You can describe any table using:

```
describe table-name
```

See the `movielens` lab in the Fusion Spark Bootcamp *(https://github.com/lucidworks/fusion-spark-bootcamp)* for a complete example of working with the Fusion Catalog API and Fusion SQL service. Also read about the *Catalog API*.

Hive configuration

Behind the scenes, the Fusion SQL service is based on Hive. Use the `/app/hive-site.xml` file to configure Hive settings.

If you change `hive-site.xml`, you must restart the Fusion SQL service with `./sql restart` (on Unix) or `sql.cmd restart` (on Windows).

Query Data Flow

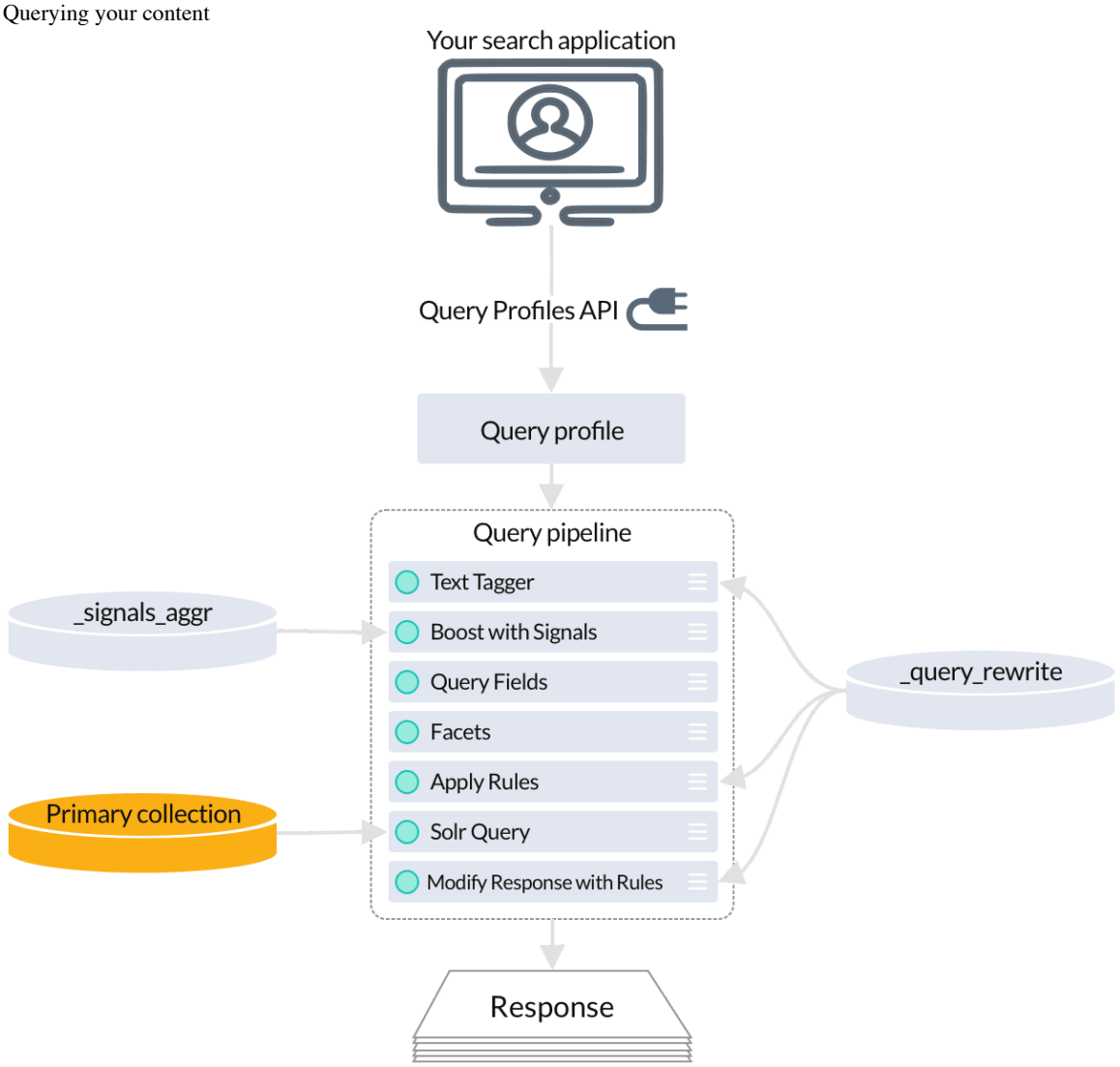
Indexing Data Flow shows you how your content, signals, and recommendations are indexed in Fusion. This topic explains how to query your indexed data.

1. Query your content

When you query your content using the default query pipeline *and* you have *signals enabled*, you also get *query rewrites* and *automatic boosting* to enhance the relevancy of your search results. In addition to matches from your primary collection, you'll get results produced by Fusion's machine learning jobs, based on *signals* from your users.

Query profiles give you a stable endpoint that can be associated with any *query pipeline*. Just specify the query profile when sending queries to the *Query API* endpoint. You can always reconfigure the query profile to point to a different pipeline, or to an *experiment*.

Querying your content

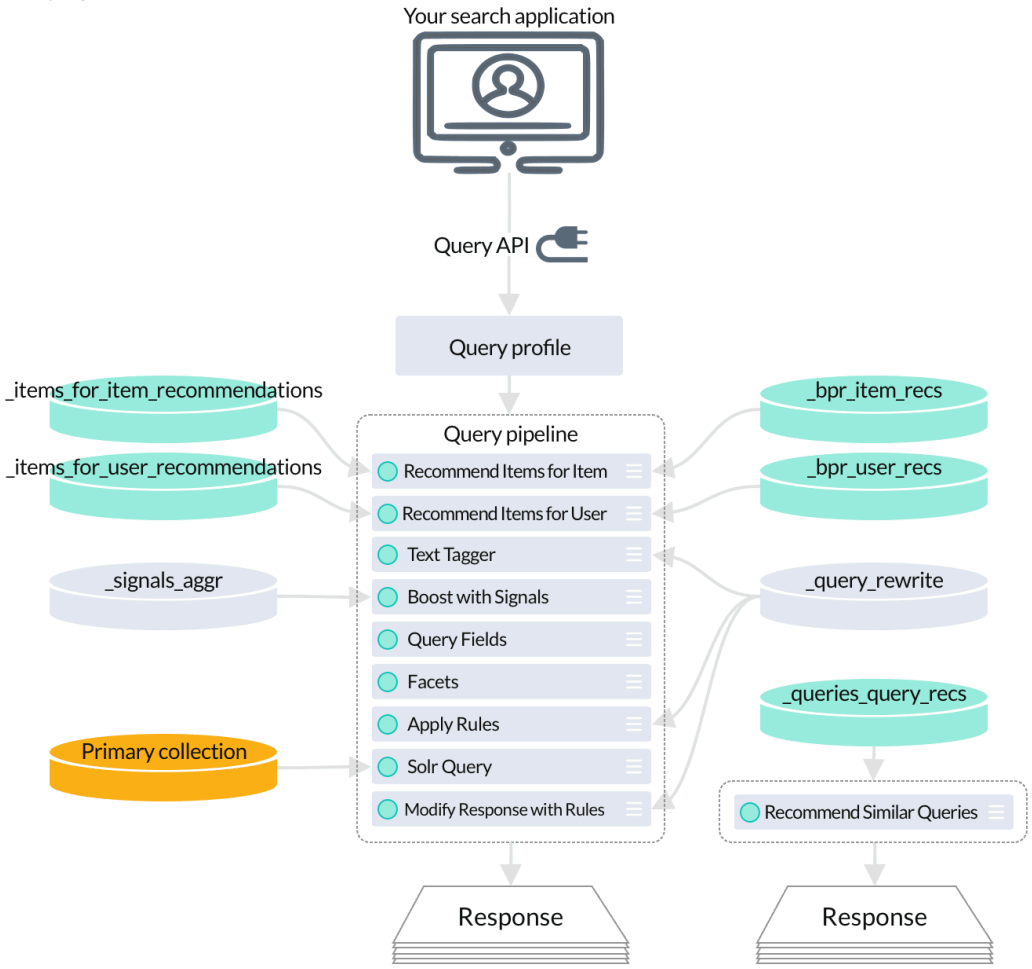


- Several stages perform *query rewriting* and *response rewriting* by reading from the `_query_rewrite` collection:
  - The *Text Tagger stage* finds *misspellings*, *phrases*, *underperforming query strings*, and *synonyms* in the incoming query.
  - The *Apply Rules stage* applies your *business rules* to the incoming query.
  - The *Modify Response with Rules stage*
- The *Boost with Signals stage* reads aggregated signals to boost search results that have been popular with users.
- The *Solr Query stage* fetches the final set of relevant search results from the primary collection.

2. Query for recommendations

To get even more relevant results based on AI-powered analysis of your signals data, your query pipeline can perform any number of secondary queries to the collections where the output from machine learning jobs is indexed. Your secondary queries can provide additional search results based on Fusion's analysis of your signals.

Querying for recommendations



- Use the *Recommend Items for Item stage* to query for *collaborative items-for-item recommendations*.
- The *Recommend Items for User stage* queries for *items-for-user recommendations*.
- The *Recommend Similar Queries stage* provides *queries-for-query recommendations*.

Query Pipelines

A Query Pipeline transforms a set of inputs into a Solr query request and it can execute requests and manipulate the Solr response as well, via a set of modularized operations called *Query Stages*. The objects sent from stage to stage are Request objects and Response objects.

Fusion stores pipeline names and definitions, allowing a pipeline to be reused across applications. Pipeline definitions can be modified, so that as an application evolves, the pipelines used by that application can evolve accordingly. During application development, the Fusion UI can be used to develop and debug a Query Pipeline.

The available stage types allow setting specific parameters for the query, such as the number of results to return or the query parser to use. You can also define facets and recommendations to be returned with the results. If Access Control Lists (ACLs) are in use, you can apply a security-trimming stage to apply user access restrictions to the results.

For details about the available REST APIs, see *Query Pipelines API* and *Query Stages API*.

Search mode

When creating a query pipeline, you can select a search mode for the pipeline to use.

- `dsl` (*Domain Specific Language*) uses expressive search queries and responses via a structured, modern JSON format.
- `legacy` primarily uses Solr parameters. See the *Solr Query Language cheat sheet*.
- `all` uses both DSL and legacy search modes. This is the default value.

The default value of `all` works well for most use cases.

Default query pipelines

Fusion creates a default query pipeline when you create an app. The query pipeline has the same name as the app.

The default query pipeline has the following pre-configured stages:

- *Text Tagger*
  This stage uses the *SolrTextTagger* handler to identify known entities in the query by searching the `COLLECTION_NAME_query_rewrite` *collection*. See *Manage Collections in the Fusion UI* for more information.

For Fusion 5.x.x organizations that do *not* have a Predictive Merchandiser license, the Solr Text Tagger handler also searches the `COLLECTION_NAME_query_rewrite_staging` collection in the case of the Fusion query rewriting *Simulator*).

- *Boost with Signals*
  The Boost with Signals query pipeline stage uses *aggregated signals* to selectively boost items in the set of search results.
- *Query Fields*
  The Query Fields query pipeline stage defines common Solr query parameters for the edismax query parser *(https://cwiki.apache.org/confluence/display/solr/The+Extended+DisMax+Query+Parser)*. If using a less-than sign (<) with DisMax, it must be escaped using a backslash. An alternative to this stage is the *Additional Query Parameters stage*.
- *Field Facet*
  The Field Facet query pipeline stage is used to add a Solr Field Facet query *(https://cwiki.apache.org/confluence/display/solr/Faceting#Faceting-Field-ValueFacetingParameters)* to the search query pipeline.
- *Apply Rules*
  This stage looks up rules that have been deployed to the `COLLECTION_NAME_query_rewrite` *collection* and matches them against the query. Matching rules that perform *query rewriting* are applied at this stage, while matching rules that perform *response rewriting* are applied by the *Modify Response with Rules* stage later in the pipeline.

To trigger a rule that contains a tag, specify the tagname in the request URL of the user search app. See *Easily define triggers in tags* for more information.

- A *Solr Query*
  The Solr Query stage transforms the Fusion query pipeline *Request* object into a Solr query and sends it to Solr.
- *Modify Response with Rules*
  Most rules operate on the request, but some *rule types*, such as banner rules or redirect rules, do their work when the response comes back. The Modify Response with Rules stage applies those rules to the response. For example, a banner rule can add a banner URL to the response before returning it to the client.

Custom query pipelines

Using the *Query Workbench* or the *REST API*, you can develop custom pipelines to suit any search application. Start with any of Fusion's built-in query pipelines, then add, remove, and re-order the *pipeline stages* as needed to produce the appropriate query results.

Asynchronous query pipeline processing

Query pipeline processing performance can be improved by enabling asynchronous processing for certain stages that make requests to secondary collections, external databases, and so on. The following stages support asynchronous processing:

- *Active Directory Security Trimming*
- *Apply Rules*
- *JDBC Lookup*
- *Security Trimming*
- *Solr Subquery*

This feature uses the fork-and-join model, where any of the stages above can create a fork on the pipeline. The parallel processes are joined again using the *Merge Async Results stage* at a later point in the pipeline.

Query Pipeline Stages

A query pipeline is made up of a series of query stages that process incoming search queries.

A pipeline stage definition associates a unique ID with a set of properties. These definitions are registered with the Fusion API service and stored in ZooKeeper for re-use across pipelines and search applications.

Fusion includes a number of specialized query stages as well as a JavaScript stage that allows advanced processing via a JavaScript program.

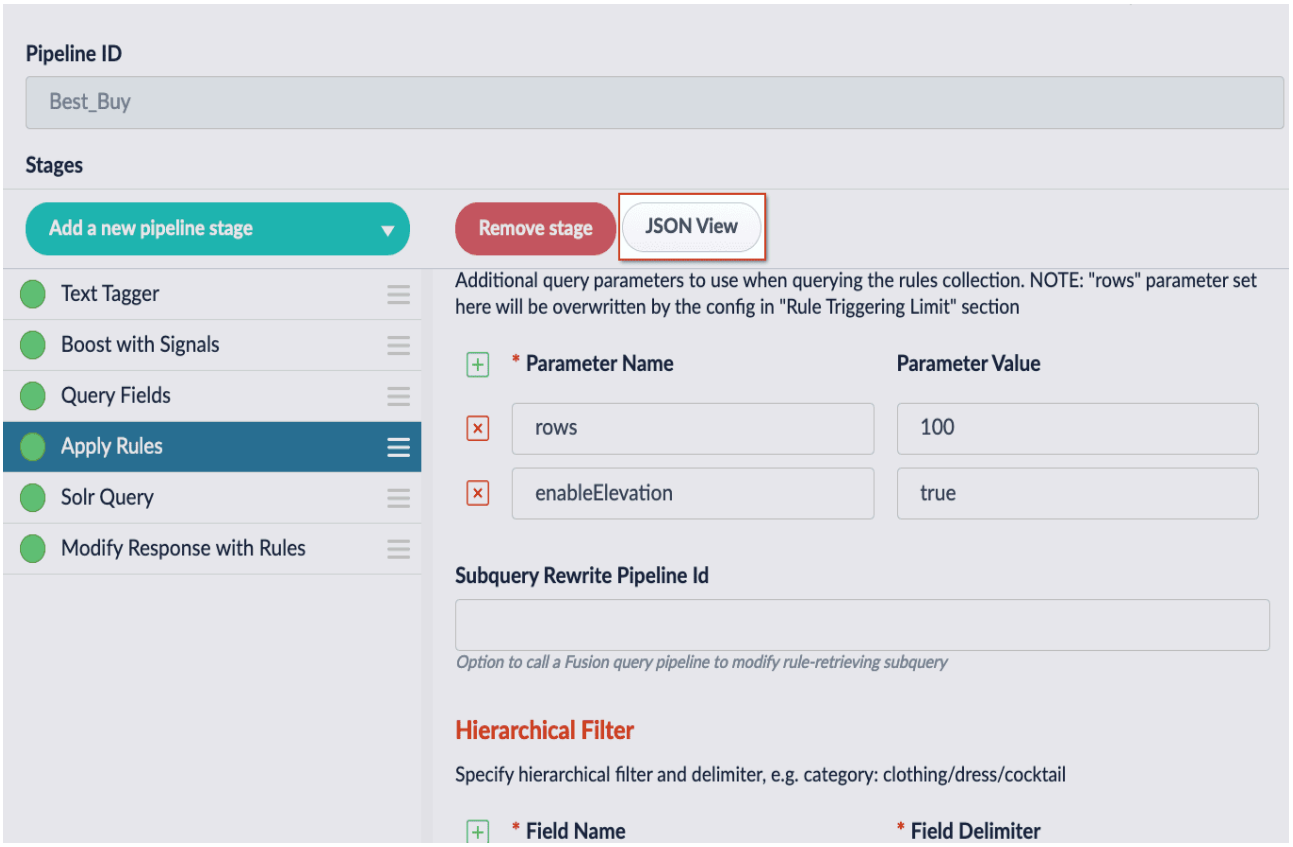
Configuring query pipeline stages

- In the Fusion UI, the *Query Workbench* provides an environment for configuring the stages in a query pipeline.
- The *Query Stages API* is used to create, list, update, or delete query stages using JSON. See also the *Query Pipelines API*.

Pipeline stage JSON editor

The pipeline stage JSON editor gives the ability to create and copy pipeline stages by pasting JSON objects in the Fusion UI. Only JSON is supported, and JSON validation is included to prevent the user from saving an invalid object.

Navigate to your pipelines, select a pipeline stage, and click the **JSON View** button to open the editor:



Existing stages are considered **READ-ONLY**. They will have only a copy button which can be used to create stages. New stages will have the option to copy or paste valid JSON. Changes are applied to the stage with the **Apply** button.

Users will still need to **Save** the stage for the changes made in the editor to be saved.
The editor has several elements worth noting:

Elements	Description
	Copies the JSON to the clipboard.
	Pastes the clipboard into the editor. Only valid JSON is accepted.
	Expands the editor to a fullscreen view.
	Condenses the editor to a compact view.
	Applies the JSON in the editor and updates the stage. The stage must be saved to preserve the changes.
	Resets changes made since the last time changes were applied.

For instructions, see *Use the Pipeline Stage JSON Editor*.

Asynchronous query pipeline processing

Query pipeline processing performance can be improved by enabling asynchronous processing for certain stages that make requests to secondary collections, external databases, and so on. The following stages support asynchronous processing:

- *Active Directory Security Trimming*
- *Apply Rules*
- *JDBC Lookup*
- *Security Trimming*
- *Solr Subquery*

This feature uses the fork-and-join model, where any of the stages above can create a fork on the pipeline. The parallel processes are joined again using the *Merge Async Results stage* at a later point in the pipeline.

Conditional query processing

Query Pipeline stages are used to modify Request objects and Response objects. Each stage can include a conditional JavaScript expression (the `condition` property in its configuration) that can access these objects.

For example, this condition first checks that the property `fusion-user-name` is specified in the Request object, then checks for a particular value:

```
request.hasParam("fusion-user-name") && request.getFirstParam("fusion-user-
name").equals("SuperUser");
```

Query Profiles

Query profiles let you consistently point your search application at a static endpoint, but give you the flexibility to change the actual query pipeline being used (and optionally, the collection and/or configuration parameters for the query pipeline).

For example, an e-commerce site might want to create a query pipeline to support a month-long promotion. After the query pipeline is configured, it can be easily enabled by changing the query profile in use by the front-end application to use the new pipeline.

To run a query through a query profile, use the REST API as described below. For CRUD operations, you can use either the REST API or the Fusion UI.

Rules Simulator query profile

> Rules Simulator is only available for Fusion 5.x.x organizations that do *not* have a valid Predictive Merchandiser or Experience Optimizer license.

The *Rules Simulator* allows product owners to experiment with rules and other query rewrites in the `COLLECTION_NAME_query_rewrite_staging` collection before deploying them to the `COLLECTION_NAME_query_rewrite` collection.

Each app has a `COLLECTION_NAME_rules_simulator` query profile, configured to use the `COLLECTION_NAME_query_rewrite_staging` collection for query rewrites instead of the `COLLECTION_NAME_query_rewrite` collection. This profile is created automatically whenever a new app is created.

See *Configure the Rules Simulator Query Profile* for more information about configuration.

Query profiles in the REST API

- *Query Profiles API* (`/query-profiles`)
  Create, read, update, and delete query profiles.
- *Query API* (`/query`)
  Run a query through a query profile by specifying the profile ID and appending the request with a *Solr query string*, as in `/api/query/<id>?<solrQuery>`.

Query profiles in the UI

To configure query profiles, click **Querying** > **Query Profiles**.

Custom JavaScript Stages for Query Pipelines

The JavaScript Query stage allows you to write custom processing logic using JavaScript to manipulate search requests and responses. The first time that the pipeline is run, Fusion compiles the JavaScript program into Java bytecode using the JDK's JavaScript engine.

The JavaScript Query stage allows you to run JavaScript functions over search requests and responses by manipulating variables called "request" and "response" which are *Request objects* and *Response objects*, respectively.

	Users who can create or modify code obtain access to the broader Fusion environment. This access can be used to create intentional or unintentional damage to Fusion.

The JavaScript Engine Used by Fusion

The JavaScript engine used by Fusion is the Nashorn engine from Oracle. See The Nashorn Java API *(https://docs.oracle.com/javase/8/docs/technotes/guides/scripting/nashorn/api.html)* for details.

JavaScript Query Stage Global Variables

JavaScript *(http://en.wikipedia.org/wiki/JavaScript)* is a lightweight scripting language. The JavaScript in a JavaScript stage is standard ECMAScript *(http://en.wikipedia.org/wiki/ECMAScript)*. What a JavaScript program can do depends on the container in which it runs. For a JavaScript Query stage, the container is a Fusion query pipeline. The following global pipeline variables are available:

Name	Type	Description
`request`	*Request for a Regular Query*	The `request` variable contains Solr query information and is referred to as a **regular** request. A regular query request does *not* include Search DSL (domain specific language) parameters.
`request`	*DSL Request*	The `request` variable is also used when the query contains parameters for a Search DSL (domain specific language) request. See *Domain Specific Language* for more information.
`response`	*Response for a Regular Query*	The `response` variable contains Solr response information, and is used to return information from a **regular** request. Because a regular request does *not* include Search DSL (domain specific language) parameters, the response will not return Search DSL results.
`response`	*DSL Response*	The `response` variable is also used to return information from a Search DSL (domain specific language) request. See *Domain Specific Language* for more information.
`ctx`	*Context*	A map that stores miscellaneous data created by each stage of the pipeline. Use the `ctx` variable instead of the deprecated `_context` global variable. The `ctx` variable is used to: • Pass data from one stage to another • Store data that needs to be passed from one custom stage to a later custom stage The data can differ between stages: • If the previous stage changes the data • Based on the configuration of each stage If the data is modified in one stage, it may cause a later stage to function irregularly.
`collection`	String	The name of the Fusion collection being indexed or queried.
`solrServer`	*BufferingsolrServer*	The Solr server instance that manages the pipeline's default Fusion collection. All indexing and query requests are done by calls to methods on this object. See solrClient *(https://lucene.apache.org/solr/5_2_1/solr-solrj/org/apache/solr/client/solrj/SolrClient.html)* for details.
`solrServerFactory`	*solrClusterComponent*	The SolrCluster server used for lookups by collection name which returns a Solr server instance for that collection, e.g. `var productsSolr = solrServerFactory.getSolrServer("products");`.
`QueryRequestAndResponse`	*QueryRequestAndResponse*	Used to create query pipeline requests and responses.
`JSONResponse`	*JSONResponse*	Returns list of info as a string.

See *Custom JavaScript Query Stage Examples* for more information.

Syntax Variants

JavaScript stages can be written using legacy syntax or function syntax. The key difference between these syntax variants is how the "global variables" are used. While using legacy syntax, these variables are used as global variables. With function syntax, however, these variables are passed as function parameters.

Legacy Syntax

Legacy syntax is used to perform very simple tasks.

```
//do some work... return doc;
```

Function Syntax

```
function(request,response) { request.addParam("foo", "bar"); }
```

Function syntax is used for the examples in this document.

Global variable `logger`

The logs are output to the query service logs for custom query stages. Access the Log Viewer and filter on this service to view the information.

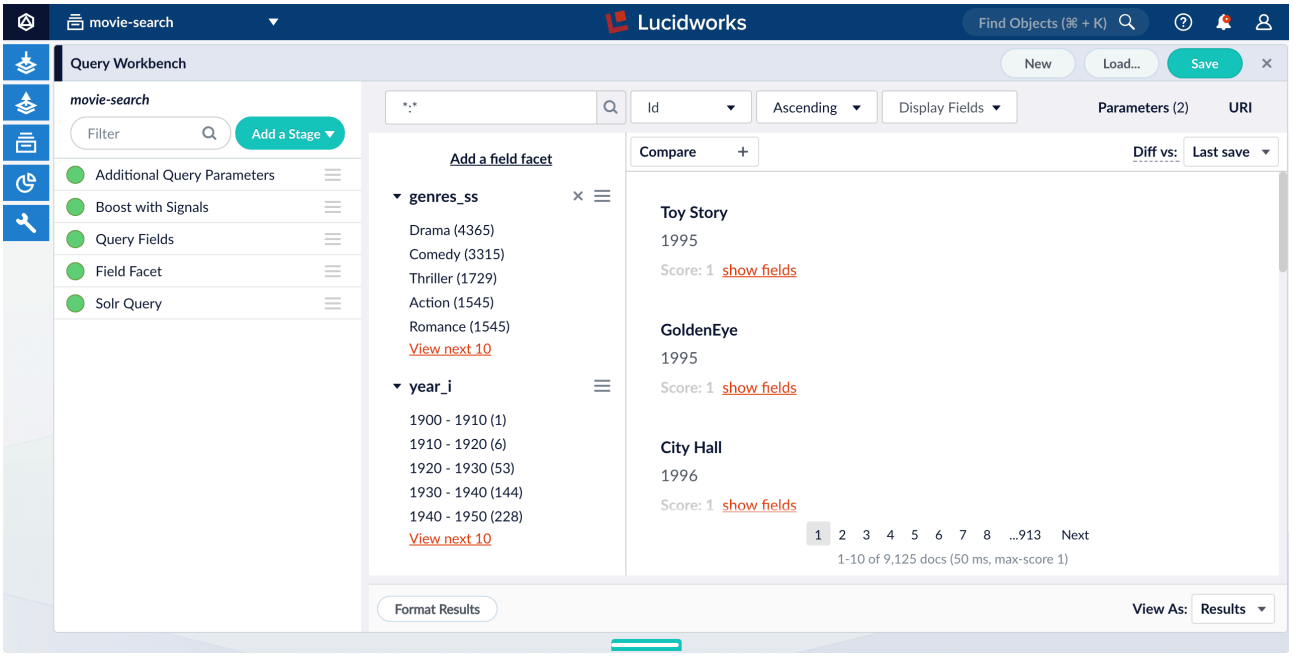
See also

- *Fusion Pipeline Javadocs*
- *JavaScript Query Stage*
- *Custom JavaScript Query Stage Examples*
- *Custom JavaScript Stages Global Variables*
- *Managed JavaScript Query Stage*
- View our training on JavaScript in Fusion *(https://academy.lucidworks.com/javascript-in-fusion)*.

The Query Workbench

The Query Workbench is where you edit query pipeline stages, results parameters, and more, then preview the impacts of the changes in real time before saving them. Relevancy tools such as boosting and blocking can be accessed and altered through the Query Workbench. The Query Workbench streamlines the process of querying indexed results and fine-tuning Query Pipelines to surface the results that best satisfy the end user's needs.

See the *Query Language Cheat Sheet* for help constructing queries.

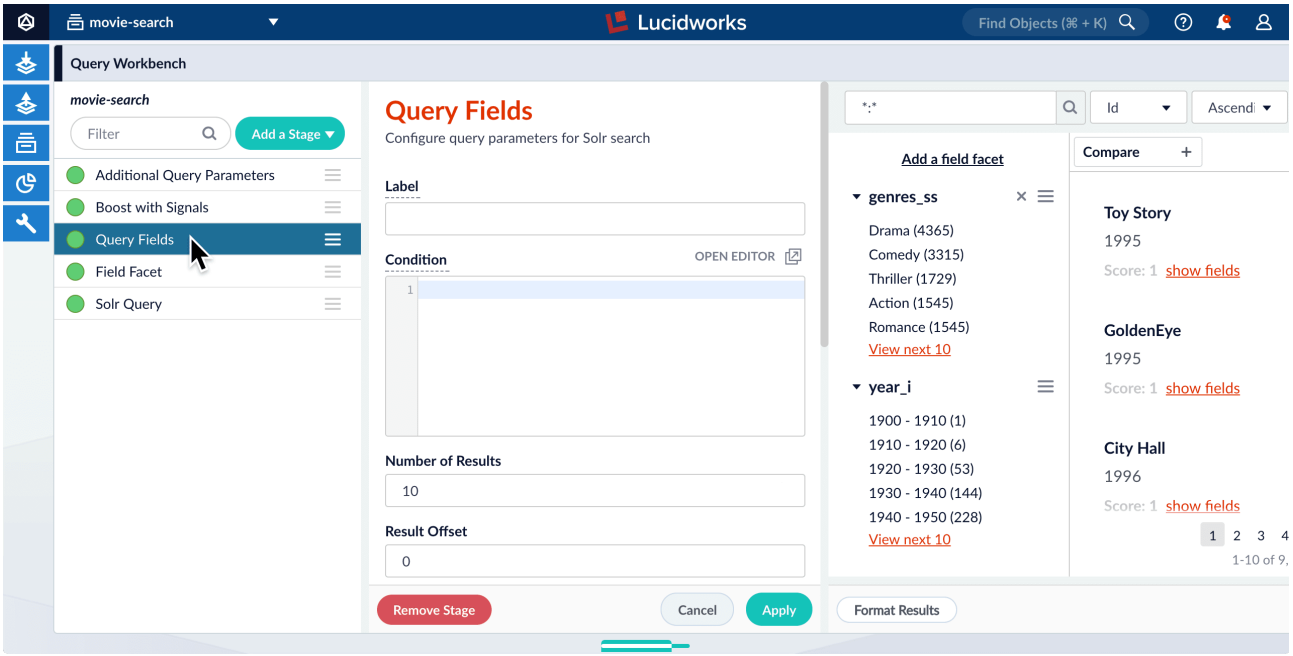


With this sandbox paradigm, you can experiment with query pipelines without consequence because Fusion uses a copy of a pipeline to simulate the new results, and it is not permanently modified in the collection until you click **Save**.

Query pipelines and pipeline stages

Query pipelines work by processing search requests and returning an ordered list of matching documents. Each pipeline consists of a series of query stages that can be added, ordered, and configured using the Query Workbench.

Select any stage in the pipeline to open its configuration panel:



Boosting and blocking

As you search your data and inspect the results, you can manipulate the rankings of individual documents. Boosting a document raises its ranking, while blocking a document removes it from search results.

Boosting and blocking affects the results for specific queries. For example, when you search for "citizen" and boost the document for "Citizen Kane", your end users will see that title boosted to the top when they also search for "citizen", but not necessarily when they search for "kane".

The Query Workbench provides convenient buttons for instantly blocking or boosting the documents that you see in the preview panel. Clicking **Boost** automatically adds the *Boost Documents stage* to your query pipeline, with a boost rule that matches your query and the document you clicked. Likewise, clicking **Block** adds and configures the *Block Documents stage*.

Compare mode

Compare mode allows you to compare search results using two different query pipelines, or compare search results from the original query pipeline and the copy being modified in the Query Workbench. On the right is the working pipeline, which you can edit. On the left is one of your saved pipelines.

Solr Query Language Cheat Sheet

This cheat sheet is a quick reference to the Solr query language. Use this syntax when querying Fusion via the *Query API*.

Additional information about Solr query syntax can be found in the Solr Query *(https://solr.apache.org/guide/8_7/the-standard-query-parser.html#specifying-terms-for-the-standard-query-parser)* documentation.

There are two ways to query a Fusion collection using the parameters below:

- Enter query parameters in the *Query Workbench* or the Quickstart.
- Append query parameters to the `/query/{id}` *endpoint*.

Wildcards

Wildcards and regular expressions are supported.

- `?`
  Single-character wildcard
- `*`
  Multi-character wildcard

Escape characters

Some characters in query fields must be escaped using a backslash `\`. Adding `\` before the following includes them in the query as literal values:

- `?`
- `*`
- `\`
- `/`
- `"`

	Queries against facets are case sensitive.

Common query parameters

`q`	Defines a query using standard query syntax *(https://solr.apache.org/guide/8_8/the-standard-query-parser.html#standard-query-parser-parameters)*. This parameter is mandatory.
`fq`	Filter query *(https://solr.apache.org/guide/8_8/common-query-parameters.html#fq-filter-query-parameter)*. A query string that limits the query results without influencing their scores.
`sort`	Sort field/direction *(https://solr.apache.org/guide/8_8/common-query-parameters.html#sort-parameter)*. The field on which to sort, followed by a space and direction (`desc` or `asc`). You can specify multiple sort fields like this: `sort=title asc,year desc`.
`rows`	Max results per page *(https://solr.apache.org/guide/8_8/common-query-parameters.html#rows-parameter)*. This sets the "page size" for paginated search results.
`start`	Pagination offset *(https://solr.apache.org/guide/8_8/common-query-parameters.html#start-parameter)*. The number of results to skip, for pagination purposes.
`fl`	Field List *(https://solr.apache.org/guide/8_8/common-query-parameters.html#fl-field-list-parameter)*. The list of fields to return in the query results.
`df`	Default field *(https://solr.apache.org/guide/8_8/the-standard-query-parser.html#standard-query-parser-parameters)*. Used to configure the `q` and `fq` parameters. If not specified, the default field is `text`.
`wt`	Response writer *(https://solr.apache.org/guide/8_8/common-query-parameters.html#wt-parameter)*. Select the response format by specifying one of Solr's response writers *(https://solr.apache.org/guide/8_8/response-writers.html)*.

Query examples

1. Search only the `title` field in the "docs" collection for the term "solr", and format the results as JSON:

```
http://FUSION_HOST:FUSION_PORT/api/query-pipelines/default/collections/docs/select?q=solr&fl=title&wt=json
```

2. Use the following query with wildcards and character escaping in the Query Workbench to search for both `BLU RAY MOVIES` and `CAMERA` classes in the `VIDEO/CD` and `PHOTO/CAMERA` departments:
   - `class_s:*RA* AND department_s:*O\/C*`
3. Use a movie collection to select the movie ID and rating where the rating is equal to 4.

```
http://FUSION_HOST:FUSION_PORT/api/query-pipelines/default/collections/docs/select?q=*:*&fl=movie_id,rating&fq=rating:4
```

4. Search for movies with "black" in the title, starting at the 20th result for pagination, then return the results as XML:

```
http://FUSION_HOST:FUSION_PORT/api/query-pipelines/default/collections/docs/select?q=black&fl=title&start=20&rows=20&wt=xml
```

Domain Specific Language (DSL)

Fusion Domain Specific Language (DSL) provides expected search results as a JSON response in a way that reduces search query complexity for the user.

Previously, users needed to understand complex syntax to express certain search queries in Fusion (for example, the best way to express a facet filter). DSL gives Fusion control over how to execute a query by transforming a structured query input into a Solr request, where we can add intelligence around the index and the user's intent.

Why Search DSL

Fusion Search Domain Specific Language (DSL) reduces search query complexity for the user. Fusion Search DSL supports expressive search queries and responses via a structured, modern JSON format.

Search DSL is an alternative to the current (now referred to as) "Legacy" format for performing search queries. Previously, users needed to understand complex syntax to express certain search queries in Fusion (for example, the best way to express a facet filter).

Compared to the legacy Solr parameter format, Search DSL is structured to more closely align with the central concepts of Fusion and provide a more usable alternative for expressing complex Fusion queries.

Search DSL gives Fusion control over how to execute a query by transforming a structured query input into a Solr request, at which time Fusion can add intelligence around the index and the user's intent.

Limitations

Despite the advantages, there are still some important limitations to be aware of when deciding whether to use the Fusion Search DSL.

	These limitations only apply to DSL queries. Legacy queries can still be issued separately from DSL queries without being subject to these limitations.

1. **Head / Tail Rewrite**. Support is currently limited when using DSL queries:
   a. In order to work on DSL queries, the `Improved Query` must be entered in as a JSON string representing the desired `main` query that should be issued (this replaces the `queryDefinition.main` field in the rewritten DSL query)
   b. The query rewrites produced by the built-in head/tail rewrite job will NOT work on DSL queries, as the job only outputs legacy-style rewrites
2. **Query Stage Support**. All query stages are fully supported
3. **Rules Support**. All kinds of rules are fully supported with the exceptions noted below
   a. **Set Params**. Not supported
   b. **Custom Rule**. Not supported

Request Format

This section describes each top-level field of a Search DSL request.

Overview

The Search DSL format is supported in the following endpoints:

query pipelines	`query-pipelines/QUERY_PIPELINE/collections/COLLECTION_ID`
query profiles	`query/QUERY_PROFILE`
templating render	`templating/renderDSL/APP_NAME`

In all cases, it should be a POST with a `Content-Type: application/json` header.

The below table briefly summarizes the function of each of the top-level fields.

Field	Description
`queryDefinition`	Defines the logic of what to query for. The `userQuery` holds the raw user query (if there is one) which is used by various Fusion components such as Rules to determine which rewrites / rules trigger. The `main` and `filter` fields accept a variety of different possible query types (terms and singleTerm are shown here) and together define the matching criteria. Additional fields are also possible here for boosting and grouping - `boostsByValues`, `boostsByQuery`, `groupedQuery`, `groupedFilters`.
`results`	Defines how the results should be displayed and organized. A variety of fields are available here, such as `cursorMark`, `start`, and `size` for pagination, sort for sorting, fields for the list of fields to display for each document, highlight for highlighting features, and group for grouping.
`facets`	The faceting configuration for the query. Defines the fields to perform faceting on as well as the desired behavior of the returned facet values. Supports range facets in the `ranges` field and field facets in the `fields` field.
`context`	Accepts parameters used by some query stages as well as DSL hints. Should not be necessary for typical use cases. For example, `tags` can be provided here to adjust which rules should trigger based on their tags, and `lw.rules.debug` can be specified with a value of "1" to include rule triggering debug info in the response.
`params`	Allows arbitrary query parameters to be added to the underlying Solr query. Should not be necessary for most use cases - the other DSL fields should be used when possible, but this field can be used when those fields do not suffice. If the Security Trimming Stage is in use, this field can be used to supply the various parameters for that stage (username, user identity key, collection, shards, etc…).

The following sections provide an example of each field in use.

	More detailed information on each field can be found in the *Fusion API Javadocs*.

queryDefinition

```
{ "queryDefinition": { "userQuery": "cyberpunk novels", "main": { "type": "terms",
"field": "body_t", "values": ["cyberpunk", "novels"], "method": "booleanQuery" },
"filters": [ { "type": "singleTerm", "field": "category_t", "value": ["books"] } ]
} }
```

The `queryDefinition` defines the logic of what to query for. The above example queries for "cyberpunk novels" against the `body_t` field, additionally filtering to only show matches that have "books" in the `category_t` field.

results

```
{ "results": { "start": 20, "size": 20, "group": { "field": "author", "size": 5 } }
}
```

The `results` defines how the results should be displayed and organized. The above example is set to show the next 20 results starting from the 21st matched document (as with Solr, `start` is 0-based). Additionally, results will be grouped by author, showing 5 results per group. In the DSL response, the head document of each group will show up in the results list with the other documents in the group in the head doc's `groupedDocs` field.

facets

```
{ "facets": { "fields": [ { "field": "category", "limit": 5 }, { "field": "brand",
"offset": "10" } ], "ranges": [ { "field": "published_dt", "gap": "+1YEAR",
"start": "2006-01-01T00:00:00Z", "end": "2020-01-01T00:00:00Z" } ] } }
```

The `facets` field controls faceting for the query. In the above example, faceting is enabled on the `category` field (limiting to show only 5 values), brand field (showing all values starting from the 11th value), and `published_dt` field (showing 1 year increments between 2006 and 2020).

context

```
{ "context": { "tags": "sometag", "lw.rules.debug": "1" } }
```

The `context` field accepts parameters used by various query stages as well as DSL hints. In the above example, the tags parameter has been specified for the Apply Rules stage which will cause only rules with the "sometag" tag to be triggered. The `lw.rules.debug` parameter has also been specified to return extra rule triggering information in the response.

params

```
{ "params": { "uid": "je985" } }
```

The `params` field allows arbitrary query parameters to be added to the underlying Solr query and is also used to supply Security Trimming Stage parameters. In the above query, the user's id has been supplied for the Security Trimming Stage.

Response Format

This section describes each top-level field of a Search DSL response.

Overview

The below table briefly summarizes the function of each of the top-level fields. The following sections provide an example of each field in a real response (some parts omitted for brevity).

Field	Description
`results`	Holds the results list with pagination info, scoring, and hit count. When grouping is performed via the `results.group` setting in the DSL request, the groups show up here with the head doc at the top level and the other docs in the group in the `groupedDocs` field.
`facets`	Holds the returned facets and facet values. Each facet has an associated `label` which can serve as the user-friendly display name and can be defined using the Set Facets rule. As this is a map, the order of facets should not be relied upon. Facet ordering can be determined instead using the Set Facets rule and the `responseValues.facet_labels` field. The `groupFacets` flag controls controls the level on which facets are calculated: before grouping (`groupFacets=false`) or after grouping (`groupFacets=true`). Default value is `true`.
`rules`	Holds data returned by any rules that triggered. There are array fields for data from particular rule types: `redirects` for Redirect, `banners` for Banner, and `jsonBlobs` for JSON Blob (organized into separate arrays based on blob type). The `responseValues` field holds values set by the Response Value rules or any rules that provide a response values setting, as well as the `facet_labels` from the Set Facets rule. When multiple rules of a given kind trigger, the one with the higher precedence appears first in the corresponding list (breaking ties on creation date, newer rule comes first).
`spellcheck`	Holds all the spelling suggestions. Suggestions are provided for any misspelled words via the `wordSuggestions` field, and for the entire query via the `querySuggestions` field.
`meta`	Assorted metadata about the query, such as timing and debug information. The `debug.solrParams` field provides a list of the query parameters that were sent in the underlying Solr query. The `timing.mainQuery` provides the QTime of the underlying Solr query, while the `timing.total` field measures the entire DSL query (both in milliseconds). The timings of individual pipeline stages are broken down in the `timing.pipeline` field.

results

```
{ "results": { "list": { "hits": 1024, "maxScore": 1.0, "pagination": { "start": 10
} "docs": [ { "collection": "books", "type": "generic", "id": "281", "score": 1.0,
"fields": { "author": "Iain Banks" "title": "Player of Games" }, "groupedDocs": {
"hits": 228, "docs": [ { "collection": "books", "type": "generic", "id": "283",
"score": 1.0, "fields": { "author": "Iain Banks" "title": "Consider Phlebas" } } ]
} }, ...additional docs... ] } } }
```

The `results` field holds the results list with pagination info, scoring, and hit count. In the above, grouping has been performed on the `author` field.

facets

```
{ "facets": { "field": { "category": { "label": "Category", "counts": [ { "name":
"Sci-Fi", "count": 7 }, { "name": "Fantasy", "count": 3 } ...additional values... ]
}, ... additional field facets... } }, "range": { "published_dt": { "label":
"Publication Date", "gap": "+1YEAR", "start": "2006-01-01T00:00:00Z", "end": "2020-
01-01T00:00:00Z", "counts": [ ...range values... ] } } }
```

The `facets` field holds the returned facets and facet values. In the above, we see facet results for the `category` field and a range facet on the `published_dt` field, each of which has a corresponding `label` configured via a Set Facets rule for a user-friendly display name for the facet.

rules

```
{ "rules": { "responseValues": { "facet_labels": [
"category:Category,published_dt:Publication Date" ] }, "jsonBlobs": { "default": [
{ "someField": "someValue" }, { "someField": "someValue2" } ] } } }
```

The `rules` field holds data returned by any rules that triggered. In the above, 3 rules have triggered. A Set Facets rule created a facet_labels entry defining the desired ordering and labels for the facets (the labels are also present in the `facets` section of the response). Two JSON Blob rules of type `default` have triggered, the first blob in the list having a higher precedence value than the second (and thus appearing first).

spellcheck

```
{ "spellcheck" : { "correctlySpelled" : false, "wordSuggestions" : { "whiskeys" : {
"startOffset" : 6, "endOffset" : 14, "origFreq" : 0, "wordFreqList" : [ { "word" :
"whiskey", "freq" : 4764 }, { "word" : "whiskies", "freq" : 15963 }, { "word" :
"whisky", "freq" : 2206 } ] }, "scoch" : { "startOffset" : 0, "endOffset" : 5,
"origFreq" : 0, "wordFreqList" : [ { "word" : "scotch", "freq" : 1556 }, { "word" :
"scott", "freq" : 3340 }, { "word" : "stock", "freq" : 78 }, { "word" : "shock",
"freq" : 9 } ] } }, "querySuggestions" : [ { "query" : "scotch whiskey", "hits" :
6320, "corrections" : { "whiskeys" : "whiskey", "scoch" : "scotch" } } ] } }
```

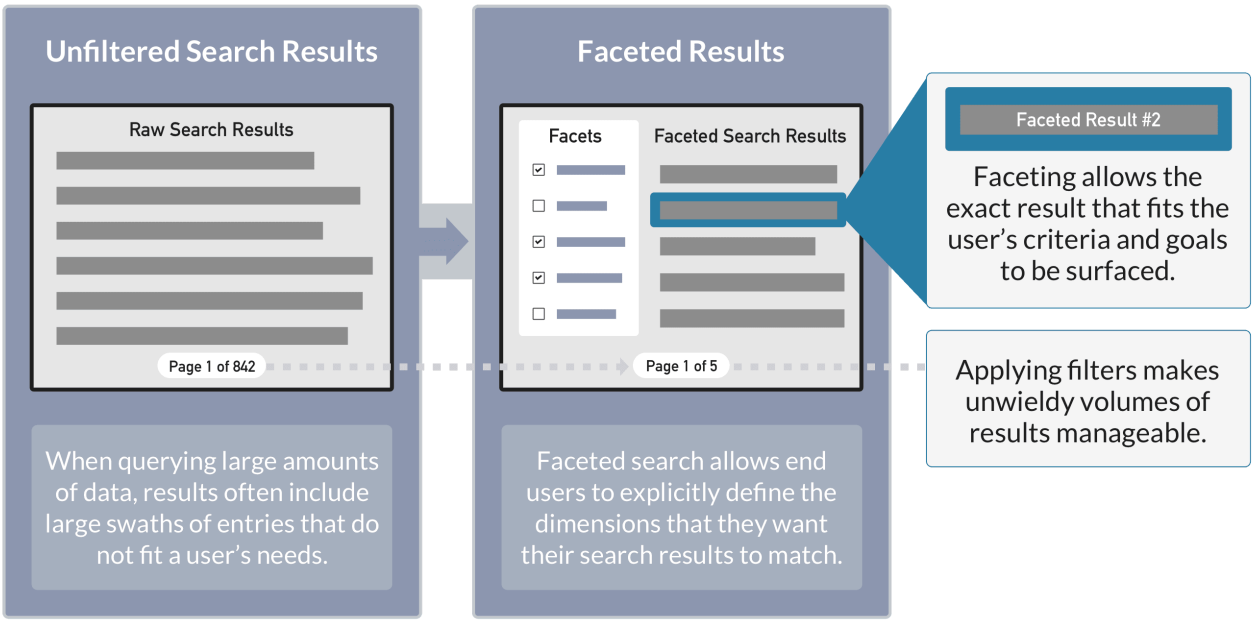
The `spellcheck` field holds all the spelling suggestions. Above, the query has misspelled “scotch whiskey” as “scoch whiskeys” and a number of suggestions are provided for each word as well as a collation with the fully corrected query.

meta

```
{ "meta": { "timing": { "total": 105, "mainQuery": 14, "pipeline": [ ... pipeline
stage timing info.. ] }, "debug": { "solrParams": { "fl": "*,score", "facet":
"true" ...additional solr params... } } } }
```

The `meta` field holds assorted metadata about the query, such as timing and debug information. Above, in the `timing.mainQuery` field we see that the underlying Solr query took 14 ms (taken from Solr’s QTime) while the `total` field shows that the entire DSL query took 105 ms. The timings of individual pipeline stages are broken down in the `pipeline` field. The `debug` field here shows the various parameters that were sent in the underlying Solr query.

Faceting



Faceting is the name given to a set of computed counts over a search result returned together with the documents which match the search query. Facets are most often used to create additional navigational controls on the search results page or panel. Users can expand and restrict their search criteria in a natural way, without having to construct complicated queries. For example, popular e-commerce facets include product category, price range, availability, and user ratings.

Fusion leverages Solr's Faceting *(https://cwiki.apache.org/confluence/display/solr/Faceting)* search components.

Field faceting

In Solr the most straightforward kind of faceting is field faceting, in which Solr's FacetComponent computes the top values for a field and returns the list of those values along with a count of the subset of documents in the search results which match that term. Field faceting works best over fields which contain a single label or set of labels from a finite, controlled lexicon such as product category. Fusion's *Facet Query Stage* can be used to configure field faceting as part of the search query pipeline.

Range faceting

Range facets are used for fields which contain date or number values. Values can be grouped into ranges by specifying additional query parameters.

To configure range faceting, use the *Additional Query Parameters Stage* to specify Solr range faceting parameters *(https://cwiki.apache.org/confluence/display/solr/Faceting#Faceting-RangeFaceting)*.

JSON faceting

In eCommerce, one product often contains multiple SKUs. In this case, it's ideal to group search results on the product ID, such that only one result for all SKUs of a given product will be returned. This is achieved by using the `unique` parameter in JSON facets *(https://solr.apache.org/guide/solr/latest/query-guide/json-facet-api.html)*. You can use JSON facets by creating Query Params in a *Query Profile* or with the *Additional Query Parameters Stage* in a Query Pipeline.

JSON faceting example

For example, if a product is available in SKUs of multiple colors, the following should apply:

1. All available colors (in any SKU in the result set) are represented in the facets
2. The `count` for those facets reflects matching product counts, not SKU counts.

In this example, eight SKU records are used across two products.

SKU	Product	Color
Sku11	Prod1	Red
Sku12	Prod1	Blue
Sku13	Prod1	Blue
Sku14	Prod1	Blue
Sku21	Prod2	Red
Sku22	Prod2	Blue
Sku23	Prod2	Green
Sku24	Prod2	Green

With the preceding data, the color facet should show:

```
Color:
Red (2)
Blue (2)
Green (1)
```

Facet stats

The Stats component *(https://solr.apache.org/guide/solr/latest/query-guide/stats-component.html)* is useful for computing statistics against fields within your document set. For example, when performing a query, you can invoke the Stats component to return information about the `mean` price of a product or determine how many documents are missing a particular field.

Facet stats example

```
https://FUSION_HOST:FUSION_PORT/api/apps/exampleapp/query/examplepipeline/select?
q=Cooking&stats=true&stats.field=price
```

```
... "stats":{ "stats_fields":{ "price":{ "min":12.34, "max":57.65, "mean":34.56,
...
```

Faceting concepts

Key Facet Concepts:

Term
: A specific value from a field.

Limit
: The maximum number of terms to be returned.

Offset
: The number of top facet values to skip in the response (just like paging through search results and choosing an offset of 51 to start on page 2 when showing 50 results per page).

Sort
: The order in which to list facet values: `count` ordering is by documents per term, descending, and `index` ordering is sorted on term values themselves.

Missing
: The number of documents in the results set which have no value for the facet field.

Choice of facet method
: (advanced) Specify Solr algorithm used to calculate facet counts. (See Facet Method Configuration *(https://cwiki.apache.org/confluence/display/solr/Faceting#Faceting-Thefacet.methodParameter)* for details). One of:

- `enum`. Small number of distinct categories.
- `fc` ("field cache"). Many different values in the field, each document has low number of values, multi-valued field.
- `fcs` ("single value string fields"). Good for rapidly changing indexes.

Further Reading

https://lucidworks.com/post/pivot-facets-inside-and-out/ *(https://lucidworks.com/post/pivot-facets-inside-and-out/)*

https://lucidworks.com/post/apache-solr-statistics/ *(https://lucidworks.com/post/apache-solr-statistics/)*

https://lucidworks.com/post/pivoting-to-the-query-using-pivot-facets-to-build-a-multi-field-suggester/
(https://lucidworks.com/post/pivoting-to-the-query-using-pivot-facets-to-build-a-multi-field-suggester/)

Additional resources

- View our training on Faceting *(https://academy.lucidworks.com/faceting-course)*.

Stored Parameters Collection

A Stored Parameters collection contains lists of parameters, where each list is has a unique identifier called a "key". These lists of parameters can be plugged into Fusion query pipelines. The Stored Parameters facility provides a mapping layer between the keys and specific actions carried during search and is somewhat similar to database triggers or stored procedures. The use for this collection is to store Solr facet parameters for a particular document or product category, where the facet parameters are stored as a JSON object. An example of what the input JSON is shown in a section below.

A Stored Parameters collection is an auxiliary Fusion collection which is associated with a primary collection via naming conventions: for a primary collection, the name of the associated Stored Parameters collection consists of the name of the primary collection plus the suffix "_stored_parameters".

In the Fusion UI, Stored Parameter collections are not displayed in the list of collections in the Collections Admin panel, nor are there controls on the UI for creating or populating these collections; all collection administration is done via calls to the REST API.

Stored Parameters for Facets

- An existing facets configuration can be converted into the required format via a once-off custom script (e.g. written by Lucidworks solution engineers or by a customer's technical team).
- Note that if a caller wants particular behavior e.g. specifying that all results should be from a given category, then it is up to them to specify that in their original request (e.g. using a filter query "fq" setting).
- A new Stored Parameter Query Stage can then be used in a query pipeline to add the required parameters for a given key (e.g. a category ID).

Stored Parameters Example

The following example shows how faceting can be carried out over a hierarchy of categories and subcategories where there are many known subcategory-specific attributes to query on. The example data is taken from the Best Buy product catalog data.

In this list of stored parameter objects, the set of fields used are:

- "id" : record ID (or "key" in the sense of a "Key-Value" store) of the category to facet on.
- "label_s" : human-readable label for this record, i.e., the category name
- "in_ids_ss" : parent categories, if an intermediate category in a category hierarchy. Intended use is in traversing a directed graph of categories.
- "out_ids_ss" : child categories, if an intermediate category in a category hierarchy. Intended use is in traversing a directed graph of categories.
- "parameters_ss" : list of arbitrary Solr parameters that can be used to construct a query that is relevant for the entity described by this record. In this example, it is the list of facet parameters to use for this specific category.

```
[ {"id": "abcat0600000", "label_s": "Movies & Music", "in_ids_ss": ["cat00000"],
"out_ids_ss": ["cat02001", "cat02015"] }, { "id": "cat02001", "label_s": "Music",
"in_ids_ss": ["abcat0600000"], "out_ids_ss": ["cat02673", "cat02012", "cat02004"]
}, { "id": "cat02010", "label_s": "Rock", "in_ids_ss": ["cat02001"], "out_ids_ss":
["cat02673", "cat02687", "cat02661", "cat02662", "cat02675", "cat02678",
"cat02680"], "parameters_ss": ["facet.field=attr_format_",
"facet.field=attr_albumLabel_", "facet.field=attr_genre_"] }, { "id": "cat02673",
"label_s": "Hard Rock", "in_ids_ss": ["cat02010"], "parameters_ss":
["facet.field=attr_format_", "facet.field=attr_albumLabel_",
"facet.field=attr_genre_"] }, { "id": "abcat0208001", "label_s": "Batteries" }, {
"id": "abcat0208002", "label_s": "Alkaline Batteries", "in_ids_ss":
["abcat0208001"], "parameters_ss": ["facet.field=attr_hardGoodType_"] } ]
```

Direct Access and Navigation

A caller can query the `COLLECTION_NAME_stored_parameters` system collection directly.

For example, a Solr request to get the human-readable label stored in field "label_s" for a specific category ID is:

```
`http://127.0.0.1:8983/solr/COLLECTION_NAME_stored_parameters/query?q=id:
<category_id>&fl=label_s`
```

Direct access allows for navigation up and down a hierarchy of categories by listing incoming and outgoing links that specify a category's location. For example, the Solr requests to find the parent(s) and children of a given category, respectively, are:

```
`http://127.0.0.1:8983/solr/COLLECTION_NAME_stored_parameters/query?q=id:
<category_id>&fl=in_ids_ss`
`http://127.0.0.1:8983/solr/COLLECTION_NAME_stored_parameters/query?q=id:
<category_id>&fl=out_ids_ss`
```

Note that this kind of "graph navigation" can become inefficient if it requires submitting a large number of queries to navigate up and down the hierarchy.

Query Enhancement

Query enhancement refers to features that improve search results by analyzing query intent, correcting malformed queries, boosting popular items, recommending items from related queries, and so on.

These topics show you how to use the basic query enhancement features:

- *Signals* are the foundation of many query enhancement strategies. Enabling and collecting signals allows you to enhance query results based on the past behavior of your users.
- Each Fusion app comes with a basic list of *stopwords* that you can modify to suit your use case.
- *Boosting* can be done manually or using signals to boost the most desirable items to the top of search results.
- *Synonyms* are alternative query terms used to expand the original query.
- *Business rules* are a versatile strategy for manually rewriting queries to address a wide range of use cases.
- *Misspelling detection* automatically rewrites misspelled queries to return relevant results.
- *Phrase detection* automatically finds phrases so that results with matching phrases can be boosted.
- *Response rewriting* modifies Solr's response, rather than the user's query.
- The *underperforming queries* feature identifies queries that can be rewritten to obtain more relevant results.
- *Recommendations* can be signals-based or content-based, and items are recommended based on the current item, user, or query.
- *Predictive Merchandiser* is a visual tool for manipulating search results to present the best items to e-commerce shoppers.
- *Experience Optimizer*, similar to Predictive Merchandiser, is a visual tool for curating search results in a knowledge management environment.

	Experience Optimizer is available in Fusion 5.4.3+.

Many of these are examples of *query rewriting*.

See also *Advanced Query Enhancement* for even more sophisticated features designed for implementation by data scientists.

Signals

A *signal* is a recorded event related to one or more documents in a collection. Signals can record any kind of event that is useful to your organization. Click signals are the most common type of signals as this is the most common action a user takes with an item. In addition, other signal types can be defined, such as "addToCart", "purchase", and so on.

Using a sufficiently large collection of signals, Fusion can automatically generate *recommendations* such as these:

- Based on the user's search query, which items are most likely to interest them?
- Based on the user's similarity to other users, which additional items are likely to interest them?

Signals are indexed in a secondary collection which is linked to the primary collection by the naming convention `<primarycollectionname>_signals`. So, if your main collection is named `products`, the associated signals collection is named `products_signals`. The signals collection is created automatically when signals are enabled for the primary collection. Signals are enabled by default whenever a new collection is created.

Signals are indexed just like ordinary documents. The signals collection can be searched like any other collection, for example by using the *Query Workbench* with the signals collection selected.

App Insights provides visualizations and reports with which to analyze your signals. App Insights mainly uses raw signals, but also uses some *aggregated* signals. Currently only the signal types Request, Response and Click are supported within the App Insights dashboards.

See *signals types and structure* for more information.

The cold start problem

The "cold start" problem means it is hard to personalize the search experience when insufficient signals have been aggregated. For example, it is hard to offer recommendations to users who have never visited before, or for queries that have never been issued before, or for items that have been recently introduced into the system.

Fusion provides solutions for this problem using its query pipelines. A query pipeline that includes stages for blocking, boosting, or recommending based on signals can also include stages that provide fallbacks. In the case where there is not enough data to provide specialized blocking, boosting, or recommendations, the pipeline can return a simpler set of search results using Solr's normal relevancy calculation.

A common solution to the cold start problem is to sort or boost on a certain field to provide pseudo-recommendations when more specific recommendations are not available. For example, you can sort on the `sales_rank` field to recommend the most popular products, or boost on the `date_added` field to recommend the newest items.

Indexing or importing Signals

Normally, signals are indexed as streaming data during the natural activity of users. See *Indexing Signals* for details.

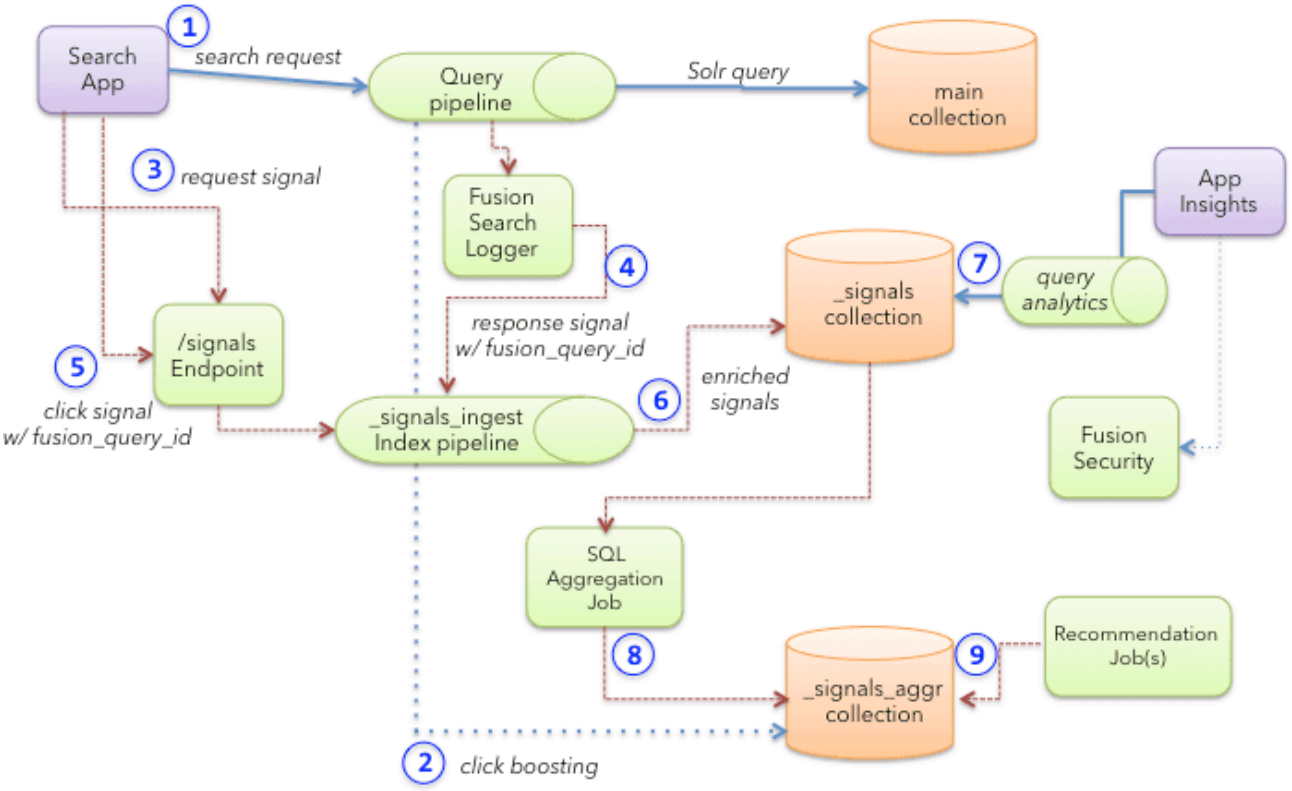
For more information, see *Parallel Bulk Loader*.

Additional resources

- View our training on Fusion Signals *(https://academy.lucidworks.com/advanced-signals)*.

Signals Data Flow

This diagram shows the flow of signals data from the search app through Fusion. The numbered steps are explained below.



1. The search app sends a query to a Fusion query pipeline.
   The query request should include a user ID and session query parameter to identify the user.
2. Optionally, the Fusion query pipeline queries the `COLLECTION_NAME_signals_aggr` collection to get boosts for the main query based on aggregated click data.
3. The search app also sends a request signal to the Fusion `/signals` *endpoint*.
   The primary intent of a request signal is to capture the raw user query and contextual information about the user's current activity in the app, such as the user agent and the page where they generated the query. The request signal does not contain any information about the results sent to Solr; it is created before a query is processed.
4. Once Solr returns the response to Fusion, the SearchLogger component indexes the complete request/response data into the `COLLECTION_NAME_signals` collection as a response signal using the `_signals_ingest` pipeline. Therefore, the response signal captures all results from Fusion as it related to the original query.

   > This is a departure from pre-4.0 versions of Fusion where query impressions were logged in a separate `COLLECTION_NAME_logs` collection. Query activity is no longer indexed into the `_logs` collection. All response signals use the `fusion_query_id` (see below) as the unique document ID in Solr.

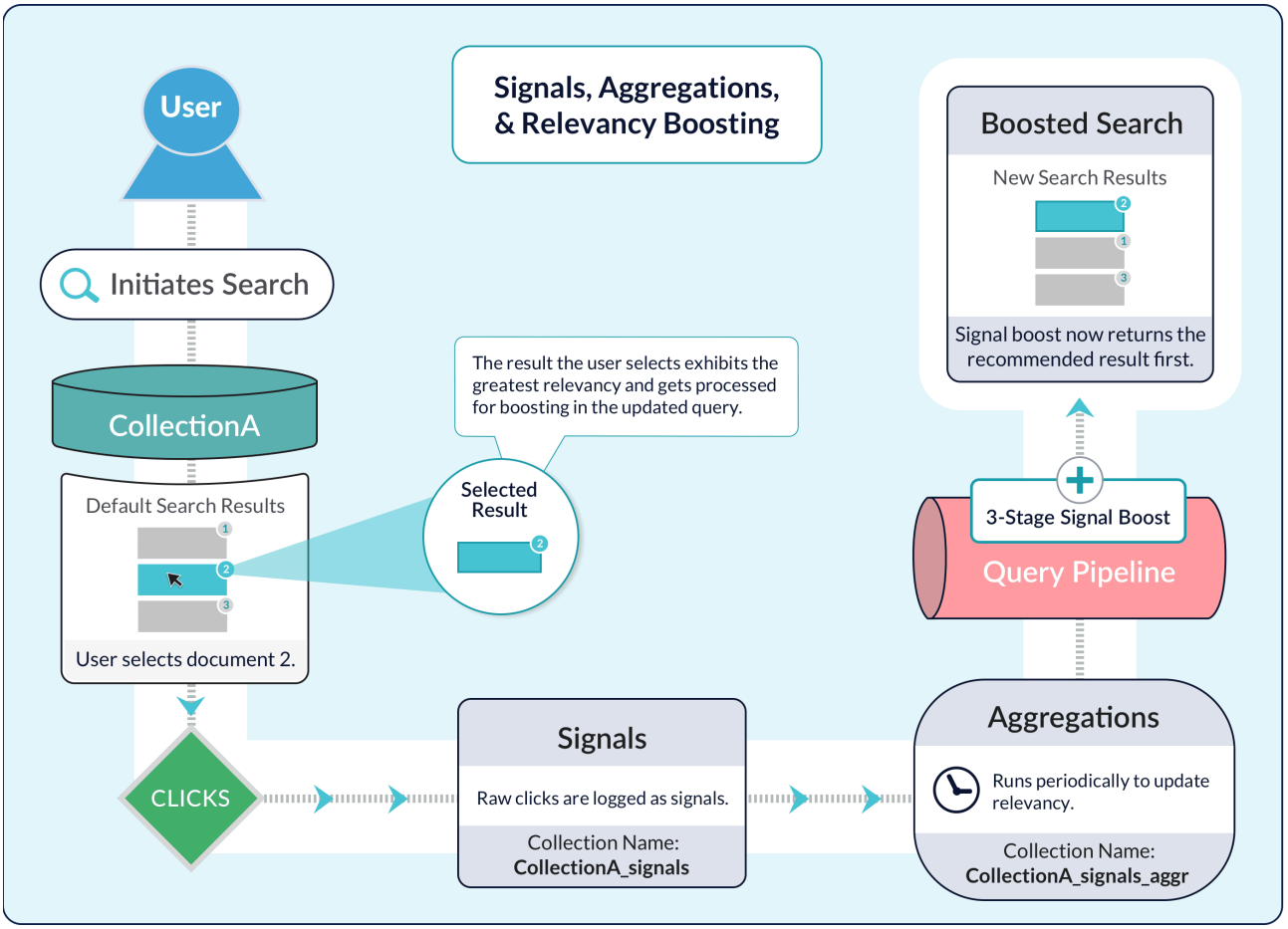
5. When the user clicks a link in the search results, the search app sends a click event to the Fusion signals endpoint (which invokes the `_signals_ingest` pipeline behind the scenes).
   The click signal *must* include a field named `fusion_query_id` in the `params` object of the raw click signal. The `fusion_query_id` field is returned in the query response (from step 1) in a response header named `x-fusion-query-id`. This allows Fusion to associate a click signal with the response signal generated in step 4. The `fusion_query_id` is also used by Fusion to associate click signals with *experiments*. For experiments to work, each click signal must contain the corresponding `fusion_query_id` that produced the document/item that was clicked.
6. The `_signals_ingest` pipeline enriches signals before indexing into the `COLLECTION_NAME_signals` collection.
   This enrichment includes field mapping, geolocation resolution, and updating the `has_clicks` flag to "true" on request signals when the first click signal is encountered for a given request using the *Update Related Document index stage*.
7. Fusion queries the `COLLECTION_NAME_signals` collection through a Fusion query pipeline to generate query analytics reports from raw signals.
8. Behind the scenes, the *SQL aggregation* framework aggregates click signals to compute a weight for each query + `doc_id` + filters group.
   The resulting metrics are saved to the `COLLECTION_NAME_signals_aggr` collection to generate boosts on queries to the main collection (step 2 above).
9. Recommendations also use aggregated documents in the `COLLECTION_NAME_signals_aggr` collection to build a *collaborative filtering-based recommender model*.

Aggregations

Aggregations compile *Signals* into a set of summaries that you can use to enrich the search experience through *recommendations and boosting*.

Fusion uses SQL to produce these signal summaries. SQL is a familiar query language that is well suited to data aggregation. Fusion's SQL Aggregation Engine is powerful and flexible.

You can perform a SQL aggregation on a signals collection for a datasource (or on some other collection) through the Fusion UI or using a Fusion API. For more information, see *Create and Run a SQL Aggregation Job*.



Rollup SQL

Most aggregation jobs run with the catch-up flag set to `true`, which means that Fusion only computes aggregations for new signals that have arrived *since* the last time the job was run, and *up to and including* `ref_time`, which is usually the run time of the current job. Fusion must "roll up" the newly aggregated rows into any existing aggregated rows in the `COLLECTION_NAME_aggr` collection.

If you are using SQL to do aggregation but have not supplied a custom rollup SQL, Fusion generates a basic rollup SQL script automatically by consulting the schema of the aggregated documents. If your rollup logic is complex, you can provide a custom rollup SQL script.

Fusion's basic rollup is a SUM of numeric types (long, integer, double, and float) and can support `time_decay` for the weight field. If you are not using `time_decay` in your weight calculations, then the weight is calculated using `weight_d`. If you do include `time_decay` with your weight calculations, then the weight is calculated as a combination of timestamp, `halfLife`, `ref_time`, and `weight_d`.

The basic rollup SQL can be grouped by `DOC_ID`, `QUERY`, `USER_ID`, and `FILTERS`. The last GROUP BY field in the main SQL is used so it will ultimately group any newly aggregated rows with existing rows.

This is an example of a rollup query:

```
SELECT query_s, doc_id_s, time_decay(1, timestamp_tdt, "30 days", ref_time,
weight_d) AS weight_d , SUM(aggr_count_i) AS aggr_count_i FROM
`commerce_signals_aggr` GROUP BY query_s, doc_id_s
```

Time-range filtering

When Fusion rolls up new data into an aggregation, time-range filtering lets you ensure that Fusion does not aggregate the same data over and over again.

Fusion applies a time-range filter when loading rows from Solr, before executing the aggregation SQL statement. In other words, the SQL executes over rows that are already filtered by the appropriate time range for the aggregation job.

Notice that the examples *Perform the Default SQL Aggregation* and *Use Different Weights Based on Signal Types* do not include a time-range filter. Fusion computes the time-range filter automatically as follows:

- If the catch-up flag is set to `true`, Fusion uses the last time the job was run and `ref_time` (which you typically set to the current time). This is equivalent to the WHERE clause `WHERE time > last_run_time AND time <= ref_time`.
- If the catch-up flag is not set to `true`, Fusion uses a filter with `ref_time` (and no start time). This is equivalent to the WHERE clause `WHERE time <= ref_time`.

The built-in time logic should suffice for most use cases. You can set the time range filter to **TO** and specify a WHERE clause filter to achieve more complex time based filtering.

Time range in Fusion is equivalent to date range in Solr *(https://solr.apache.org/guide/working-with-dates.html)*.

Example values for time range:

- `[* TO NOW]` - all past events
- `[2000-11-01 TO 2020-12-01]` – specify by exact date
- `[* TO 2020-12-01]` – from the first data point to the end of the day specified
- `[2000 TO 2020]` - from the start of a year to the end of another year

SQL functions

A Spark SQL aggregation query can use any of the functions provided by Spark SQL *(https://spark.apache.org/docs/latest/api/sql/index.html)*. Use these functions to perform complex aggregations or to enrich aggregation results.

Weight aggregated values using a time-decay function

Fusion automatically uses a default `time_decay` function to compute and apply appropriate weights to aggregation groups during aggregation. Larger weights are assigned to more recent events. This reduces the impact of less-recent signals. Intuitively, older signals (and the user behavior they represent) should count less than newer signals.

If the default `time_decay` function does not meet your needs, you can modify it. The `time_decay` function is implemented as a User Defined Aggregate Function *(https://spark.apache.org/docs/latest/sql-ref-functions-udf-aggregate.html)* (UDAF).

Full function signature
This is the UDAF signature of the default `time_decay` function:

```
time_decay(count: Long, timestamp: Timestamp, halfLife: String (calendar interval),
ref_time: Timestamp, weight_d: Double)
```

Abbreviated function signature and default values
Your function call can also use this abbreviated UDAF signature, that omits `halfLife`, `ref_time`, and `weight_d`:

```
time_decay(count: Long, timestamp: Timestamp)
```

In this case, Fusion fills in these values for the omitted parameters: `halfLife` = `30 days`, `ref_time` = `NOW`, and `weight_d` = `0.1`.

> Users indexing custom signals with `weight_d` specified should ensure that the "default" value matches the `weight_d` parameter used by `time_decay`.

Matching legacy aggregation

To match the results of legacy aggregation, either use the abbreviated function signature or supply these values for the mentioned parameters: `halfLife` = `30 days`, `ref_time` = `NOW`, and `weight_d` = `0.1`.

Parameters

Parameters for `time_decay` are:

Parameter	Description
`count`	Number of occurrences of the event. Typically, the increment is 1, though there is no reason it could not be some other number. In most cases, you pass `count_i`, which is the event count field used by Fusion signals, as shown in the *SQL aggregation examples*.
`timestamp`	The date-and-time for the event. This time is the beginning of the interval used to calculate the time-based decay factor.
`halfLife`	Half life for the exponential decay that Fusion calculates. It is some interval of time, for example, `30 days` or `10 minutes`. The `interval` prefix is optional. Fusion treats `30 days` as equivalent to `interval 30 days`.
`ref_time`	Reference time used to compute the age of an event for the time-decay computation. It is usually the time when the aggregation job runs (`NOW`). The reference time is not present in the data; Fusion determines the reference time at runtime. Fusion automatically attaches a `ref_time` column to every row before executing the SQL.
`weight_d`	Initial weight for an event if not specified in the signal, prior to the decay calculation. This value is typically not present in the signal data. If some signals *do* contain `weight_d` values, this parameter should be set to match the "neutral" value there. You can use SQL to compute `weight_d`; see *Use Different Weights Based on Signal Types* for an example.

Sample calculation of the age of a signal

This is an example of how Fusion calculates the age of a signal:

Imagine a SQL aggregation job that runs at `Tuesday, July 11, 2017 1:00:00 AM (1499734800)`. For a signal with the timestamp `Tuesday, July 11, 2017 12:00:00 AM (1499731200)`, the age of the signal in relation to the reference time is 1 hour.

Signals Types and Structures

Signals can be broadly categorized as *implicit or explicit*. When signals are enabled, Fusion produces several *built-in signal types* by default, all of which are implicit signals. You can also create *custom signal types*, including explicit signals. Be sure to verify that your signals include all of the *important fields* for best results. It is also useful to *rank your signal types* in terms of how strongly each type indicates a user's interest in an item.

Implicit signals vs explicit signals

Signals can reveal a user's level of interest in an item in two main ways:

- Implicit
  The user shows interest by engaging with the item/document through clicks, searches, and so on. Since this type of interaction requires no additional effort on the user's part, these types of signals tend to be plentiful. They can be used to infer a measurable value of interest in order to build an accurate recommender system.
- Explicit
  An explicit signal is created when a user intentionally assigns a clear, measurable value to an item, such as by giving it a rating. This value can be used to rank items, for example. Since this requires the user to invest extra time to provide the information, the number of ratings tends to be small compared to the total number of users interacting with the item.

You can create *recommendations* based on implicit signals out of the box. For recommenders based on explicit signals, contact your Lucidworks Professional Services representative.

Built-in signal types

There are five built-in signal types:

annotation	Annotation signals are generated when a user bookmarks, likes, or comments on a document. Annotation signals are likewise generated when the user removes a bookmark, like, or comment. Annotation signals are generated by *App Studio*. If you are not using App Studio, this type of signal is not relevant to your search application.
login	Login signals record information about specific users when they log in to an application. This includes a time stamp and various session details.
request	A request signal is generated by a front-end search app and captures the raw user query and other contextual information about a user and their journey through the search app.
response	Response signals are automatically generated by a query pipeline when the signals feature is enabled for a collection.
click	Click signals are generated when a user clicks on a page element that is being monitored by the search app. Click signals are sent from the search app to Fusion.

Because response signals and their fields are automatically generated, this topic does not cover what response signal fields are required.

Annotation signals

Annotation signals are generated when a user bookmarks, likes, or comments on a document. Annotation signals are likewise generated when the user removes a bookmark, like, or comment.

Annotation signals are generated by *App Studio*. If you are not using App Studio, this type of signal is not relevant to your search application.

Login signals

Login signals record information about specific users when they log in to an application. This includes a time stamp and various session details.

Request signals

A request signal is generated by a front-end search app and captures the raw user query and other contextual information about a user and their journey through the search app. The request signal contains no information about the documents the user searched for.

A request signal should have the following fields:

```
[ { "id":"288fe4f7-6680-403e-8d18-27647cdd9989", "timestamp":1518717749409,
"type":"request", "params":{ "user_id":"admin", "session":"ef4e00cd-91bb-45b4-be80-
e81f9f9c5b27", "query":"USER QUERY HERE", "app_id":"SEARCH APP ID",
"ip_address":"0:0:0:0:0:0:0:1", "host":"Lucids-MacBook-Pro-5.local", "filter":[
"field1/value", ... ], "filter_field":[ "field1" ] } } ]
```

Additional optional fields are used by *App Insights*. In the raw signal, optional fields should be inside the `params` object. Optional fields are as follows:

```
"page_title":"Fusion Search", "path":"/search", "browser_type":"Browser",
"browser_version":"64.0.3282.140", "browser_name":"Chrome",
"referrer":"http://localhost:8080/", "ctx_prev_uri":"/", "ctx_prev_query":"",
"ctx_prev_path":"/", "os_manufacturer":"Apple Inc.", "os_name":"Mac OS X",
"os_id":"778", "os_device":"Computer", "os_group":"Mac OS X"
```

Response signals

Response signals are automatically generated when a query pipeline receives a search request and that request is processed. This occurs only when the signals feature is enabled for a collection.

	Front-end search applications should not send response signals to Fusion directly, as those would conflict with the auto-generated signals.

Response signals capture much of the same information as a request signal. Response signals also capture information about the actual results returned to the front-end search application, such as the search query, the query state (including sorts, pagination, and applied filters) and all the results from Fusion that are related to the query. Response signals can also identify search requests that returned no results to the user or requests that resulted in no clicks from the user.

Fusion creates a unique `fusion-query-id` to each response signal. This is used to correlate the downstream click signals with the original search request. After a user receives the results from their search, they can interact with the documents. In order to know which results (if any) were shown to the user, we use the response signals correlated with the unique `fusion-query-id`.

A response signal has the following explicit fields, plus any additional query parameters sent by the search application for a query:

Field Name	Description	Example
`id`	The x-fusion-query-id generated by the query-pipeline used for associating click signals with queries in experiments and aggregation jobs.	`TwWCn3Dz`
`type`	Signal type	`response`
`response_type`	Used by Insights to determine if this query had results or was empty	`results | empty`
`session`	User session ID; the search app should pass the session ID in the query params for a query	`UUID`
`query`	The actual query string sent to Solr from Fusion	`ipad`
`query_orig_s`	The incoming query from the search app before it is enriched by the query pipeline	`ipad`
`query_id`	A hash generated from the session, query, and filters fields; used as a rollup key in Insights to group activity by a specific	`SHA1 hash`
`filters_s`	Filter queries sent to Solr; the Fusion SearchLogger component combines multiple fq parameters into a single value delimited by " $ "	`{!tag=format}format:(vhs) $ {!tag=type}type:(movie)`
`filter`	Reformatted filter queries	`field1/value`
`user_id`	User ID; the search app should pass the user_id in the query params	`admin`
`doc_ids_s`	A comma-delimited list of document IDs returned for the page of results; this field is used by Fusion Spark jobs, such as the ground truth job, to perform click/skip analysis	`123,456,789`
`pipeline_id`	Fusion query pipeline that processed this query	`_system`
`collection`	Fusion collection	`my_collection`
`qtime`	Query time from Solr, in milliseconds	`10`
`rows`	Number of rows requested for this query	`10`
`hits`	Total number of documents matching the query	`10000`
`totaltime`	Total processing time of this query in milliseconds, includes Solr qtime and Fusion query processing time	`15`
`timestamp_tdt`	Timestamp when the query request was received by Fusion	`2018-02-15T18:17:42.560Z`

Field Name	Description	Example
`res_offset`	Offset of results; this field is used by experiment metrics to calculate MRR	`0`
`res_pos`	Position of the clicked result within the list of results	`3`
`params.*`	Any other query param sent from the search app to Fusion that was not already mapped to a declared field	`params.defType_ss=edismax`

Fusion's experiment framework relies heavily on response signals and the linking between response and clicks signals using the `fusion_query_id`.

Click signals

Click signals are explicit events that capture any type of user interaction that the business is interested in keeping track of. The basic click signal records the action of a user clicking an item in a context, whether that context is within search results, category browse, type ahead suggestions, or other locations. Each unique action receives a name such as click2pdp, add2cart or purchase.

When a user clicks a search result, your search app should send a click signal to Fusion. All click signals should include a `fusion_query_id` field pulled from the query response header `x-fusion-query-id`.

In addition, click signals should include the following fields:

```
[ { "id":"SOME UUID HERE", "timestamp":1518725351750, "type":"click", "params":{
"fusion_query_id":"ABkaEA11", "user_id":"admin", "session":"b3a15101-9e30-4e28-
8a23-d1f663c2ee06", "query":"tiger woods", "ctype":"result", "res_offset":0,
"filter":[ "type/Game" ], "ip_address":"0:0:0:0:0:0:0:1", "host":"Lucids-MacBook-
Pro-5.local", "doc_id":"9502308", "app_id":"SEARCH APP ID", "res_pos":1,
"filter_field":[ "type" ] } } ]
```

Additional optional fields are used by *App Insights*. In the raw signal, optional fields should be inside the `params` object. Optional fields are as follows:

```
"browser_type":"Browser", "browser_version":"64.0.3282.140",
"browser_name":"Chrome", "referrer":"http://localhost:8080/", "ctx_prev_uri":"/",
"ctx_prev_query":"", "ctx_prev_path":"/", "os_manufacturer":"Apple Inc.",
"os_name":"Mac OS X", "os_id":"778", "os_device":"Computer", "os_group":"Mac OS X"
"url":"http://localhost:8080/#/product/9502308", "label":"Tiger Woods PGA Tour 09
All-Play - Nintendo Wii",
```

Custom signal types

The signal `type` parameter can also take arbitrary values for custom signal types. For example, these custom events are important for e-commerce sites:

- Add-to-favorites
- Add-to-cart
- Remove-from-cart
- Purchase
- Hover/quick-view

To collect custom signals, configure your front-end search application to send signals to Fusion using a custom value for the `type` field. Custom signals should also include the fields described *below* in order to get the best results from *aggregation* and

recommendation jobs.

To use custom signals in *recommendations*, you must add them to the value of the `signalTypeWeights` parameter in the configuration for the *`COLLECTION_NAME_user_item_preferences_aggregation`* job and the *`COLLECTION_NAME_user_query_history_aggregation`* job.

Custom signals can be analyzed in *App Insights* just like pre-defined signal types.

Required signal fields

Depending on how you use signals, certain fields are required. These are signals collection field names and not the JSON field names in the in-bound signals document. An example is when sending the **user id**, write it as `params.user_id`.

The fields mentioned in this section are defined as follows:

Field Name	Example Value	Description
`count_i`	1	Number of times an interaction event occurred with this item
`doc_id`	NMDDV	Product ID or Item ID
`query`	xwearabletech	A query string from the user
`session_id`	91aa66d11af44b6c90ccef44d055cf9a	Id for session in which user generated the signal
`type`	quick_view_click	Type of session the user used to interact with the platform
`user_id`	11506893	ID of user during the session
`timestamp_tdt`	2018-11-20T17:58:57.650Z	Time when signal was generated

Some *signal types*, including *custom signal types*, may include additional fields.

Required fields by use case

Aggregations, *recommendations*, and *App Insights* work best when certain fields are present in your signals. See these topics for details:

- *Required Signals for Jobs*
- *Required Signals for App Insights*

Required fields by signal type

The following table describes which fields are required for annotation, click, login, and request signals.

	Requests (or queries) can also require additional, available user data for the search.

Field	Type	Description	Example	Required
`id`	string	Unique ID for the signal.	`b0ee5307-6223-4150-ac5a-d0d8113aa480`	✅ Annotation ✅ Click ✅ Login ✅ Request
`doc_id`	string	Product ID or Item ID of the clicked result.	`NMDDV`	✘ Annotation ✅ Click ✘ Login ✘ Request
`date`	timestamp	Timestamp of when the signal was generated. This timestamp follows Unix epoch time formatting.	`1518717749409`	✅ Annotation ✅ Click ✅ Login ✅ Request
`user_id`	string	Unique ID for the user that generated the signal.	`admin`	✅ Annotation ✅ Click ✅ Login ✅ Request
`session`	string	Unique ID for the user's browser session.	`ef4e00cd-91bb-45b4-be80-e81f9f9c5b27`	✅ Annotation ✅ Click ✅ Login ✅ Request
`host`	string	Host name of the server which is hosting the app that is generating the signal.	`x.y.z`	✅ Annotation ✅ Click

Field	Type	Description	Example	Required
				✅ Login ✅ Request
`ip_address`	string	IP address of the user that generated the signal.	`80.6.99.35`	✅ Annotation ✅ Click ✅ Login ✅ Request
`app_id`	string	Name of the application that is generating the signal.	`myApp`	✅ Annotation ✅ Click ✅ Login ✅ Request
`annotation_type`	string	Type of the annotation signal, either "comment", "bookmark", or "like". Required fields for "comment": • `comment`. The comment itself. • `target`. The target that the comment refers to. Required fields for "bookmark": • `title`. The bookmark title. • `url`. The bookmark url. Required fields for "like": • `like` • `dislike`	`bookmark`	✅ Annotation ✘ Click ✘ Login ✘ Request
`query`	string	Terms of the query.	`ipad`	✘ Annotation ✅ Click ✘ Login ✅ Request
`fusion_query_id`	string	Unique ID for the `query` that is automatically generated from the Fusion response signal.	`ABkaEA11`	✘ Annotation ✅ Click

Field	Type	Description	Example	Required
				✘ Login ✘ Request
`filter`	array of string	List of filters associated with the query, which in turn is associated with signal.	`["type/tablet","category/electronics"]`	✘ Annotation ✅ Click ✘ Login ✅ Request
`ctype`	string	Type of click.	`result`	✘ Annotation ✅ Click ✘ Login ✘ Request
`res_pos`	number	Position of the clicked result within the list of results.	`3`	✘ Annotation ✅ Click ✘ Login ✘ Request
`res_offset`	number	Result page.	`2`	✘ Annotation ✅ Click ✘ Login ✘ Request
`url`	string	URL of the page that the signal originated from.	`http://localhost:8080/products/search`	✘ Annotation ✘ Click ✘ Login ✅ Request
`path`	string	URL path of the page that the signal originated from.	`/search`	✘ Annotation

Field	Type	Description	Example	Required
				✘ Click ✘ Login ✅ Request
`page_title`	string	Title of the page that the signal originated from.	`Search Page`	✘ Annotation ✘ Click ✘ Login ✅ Request

Field name suffixes

Fusion can add suffixes when fields are indexed. This table lists common suffix values.

Single Value Suffix	Multivalued Sufix	Type
`*_b`	`*_bs`	boolean
`*_d`	`*_ds`	double
`*_dt`	`*_dts`	date
`*_f`	`*_fs`	float
`*_i`	`*_ii`	int
`*_l`	`*_ls`	long
`*_s`	`*_ss`	string
`*_t`	`*_ts`	text

Signal field count analysis

Lucidworks recommends performing signal field count analysis to determine whether any of the fields above are missing from some of your signals.

The table below shows how to query for specific fields using the *Query Workbench* in order to compare the number of results for each field with the total number of documents in the signals collection. In the examples in the third column, some fields appear in all 33,477,919 signals documents, while others appear in fewer documents.

Field name	Query	Example number of documents
`ALL`	`*:*`	33,477,919
`count_i`	`count_i:[* TO *]`	11,101,165
`doc_id`	`doc_id:[* TO *]`	23,216,297
`id`	`id:[* TO *]`	33,477,919
`query`	`query:[* TO *]`	19,724,598
`session_id`	`session_id:[* TO *]`	11,101,165
`type`	`type:[* TO *]`	33,477,919
`user_id`	`user_id:[* TO *]`	26,117,399
`timestamp_tdt`	`timestamp_tdt:[* TO *]`	26,117,399

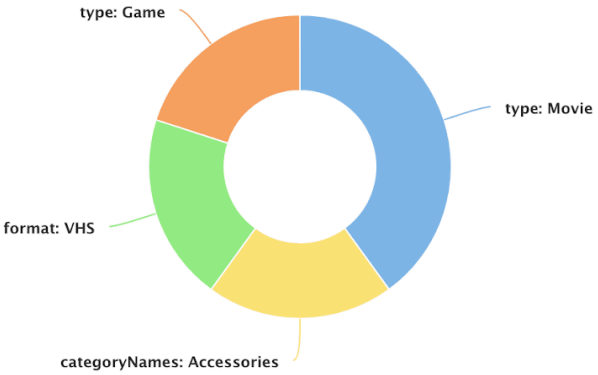
You can also get the number of signals documents that contain all of the required fields by using the following query:

```
count_i:[* TO *] doc_id:[* TO *] id:[* TO *] query:[* TO *] type:[* TO *] user_id:
[* TO *] timestamp_tdt:[* TO *] session_id:[* TO *]
```

The `query_id` field

For each incoming signal, Fusion calculates a value for the `query_id` field.

App Insights uses to create group-by-query reports like the one shown below:



> The `query_id` field should not be confused with the `fusion_query_id`, which is a unique ID for each query processed by a Fusion query pipeline, or with `query_s` which is the query string.

To calculate the value, Fusion creates a hash based on `session`, `query`, and `filter` fields, then saves it into the `query_id` field.

The `filter` field can either be passed in by the search app, or computed by the SignalFormatterStage (the first stage in the *`_signals_ingest` pipeline*) using the raw filter queries. For instance, on a response signal that is generated by a query pipeline, the

following `fq` query params get translated into the multi-valued `filter` field:

- Raw query parameters:

```
fq={!tag=format}format:(VHS)&fq={!tag=type}type:(Movie)
```

- `filters_s` field (created by the SearchLogger component):

```
{!tag=format}format:(vhs) $ {!tag=type}type:(movie)
```

- `filter` field:

```
"filter":["format/VHS", "type/Movie"]
```

App Insights uses the `filter` field to generate various reports.

Signal type ranking

When you have defined some custom fields, it is useful to rank them according to how strongly they indicate a user's interest in an item. While it's not necessary to exclude certain signal types from the main signals collection, some can be excluded from signal aggregations in order to focus on the most important fields when generating *recommendations*.

Deleting Old Signals

Signals are not automatically deleted by default, and over time they occupy an increasing amount of storage space.

To avoid running out of storage space as a result of your growing collection of signals, you must decide on a signals retention policy, then configure and schedule a *REST Call job* that regularly deletes old signals.

The duration for which signals should be kept depends on your use case and your organization's policies. For example, in some cases signals could be deleted after 30 days, while in other cases they may remain useful for a year, or even forever.

Required Signals for App Insights

Although there are many fields that are relevant to *signals*, this topic describes which fields are **required** for each *type of signal*. Without these fields, reports and dashboards may not function as expected.

Report	Signal Types Used	Required Field	Type	Description	Example
Search queries	✘ Annotation ✅ Click ✘ Login ✅ Request ✅ Response	`query`	string	Terms of the query.	`physics`
Facets used	✘ Annotation ✘ Click ✘ Login ✅ Request ✘ Response	`filter_field`	array of string	List of filters associated with the query, which in turn is associated with signal.	`["categories_s"]`
Facet filters applied	✘ Annotation ✘ Click ✘ Login ✅ Request ✘ Response	`filter`	array of string	List of **applied** filters associated with the query, which in turn is associated with signal.	`["categories_s/math.OA"]`
Application servers	✘ Annotation ✅ Click ✘ Login ✅ Request ✘ Response	`host`	string	Host name of the server that is hosting the app, which in turn is generating the signal.	`lucidworks`
Response times	✘ Annotation	`time`	number	Response time (in milliseconds).	`17`

Report	Signal Types Used	Required Field	Type	Description	Example
	✘ Click ✘ Login ✘ Request ✅ Response				
URLs clicked	✘ Annotation ✅ Click ✘ Login ✘ Request ✘ Response	`url`	string	URL of the page that was selected from the search results. This typically results from a click signal.	`http://arxiv.org/pdf/astro-ph/0611688v1`
Search pages	✘ Annotation ✘ Click ✘ Login ✅ Request ✘ Response	`page_title`	string	Title of the page that the signal originated from.	`Search Preview`
Search platforms	✘ Annotation ✅ Click ✘ Login ✅ Request ✅ Response	`platform`	string	Name of the search platform.	`fusion`
Visitor countries	✘ Annotation ✅ Click ✘ Login	`geo_country_name`	string	Originating country of the user.	`United States`

Report	Signal Types Used	Required Field	Type	Description	Example
	✅ Request ✘ Response				
Visitor cities	✘ Annotation ✅ Click ✘ Login ✅ Request ✘ Response	`geo_city_name`	string	Originating city of the user.	`San Francisco`
Browsers	✘ Annotation ✅ Click ✘ Login ✅ Request ✘ Response	`browser_name`	string	Name of the browser that generated the signal.	`Firefox`
Operating System	✘ Annotation ✅ Click ✘ Login ✅ Request ✘ Response	`os_name`	string	Name of the operating system used by the user.	`Ubuntu`
Device types	✘ Annotation ✅ Click ✘ Login ✅ Request	`os_device`	string	Type of the device used by the user.	`Computer`

Report	Signal Types Used	Required Field	Type	Description	Example
	✘ Response				
Websites people are coming from	✘ Annotation ✅ Click ✘ Login ✘ Request ✘ Response	`referrer_domain`	string	URL of the page that the signal originated from.	`http://www.google.com`
Users	✘ Annotation ✅ Click ✘ Login ✅ Request ✘ Response	`user_id`	string	Unique ID for the user that generated the signal.	`admin`
User domains	N/A	`domain`	string	Domain of the user that generated the signal.	`lucidworks.com`
Types of event	✅ Annotation ✅ Click ✅ Login ✅ Request ✅ Response	`type`	string	Type of the signal.	`click`
Head Tail analysis	✘ Annotation			This report is populated by running a head tail job in Fusion. With a collection called `foo`, for example, run the `foo_head_tail` job.	N/A
	✅ Click	type	string		N/A
	✅ Click	query	string		N/A

Report	Signal Types Used	Required Field	Type	Description	Example
	✘ Login				N/A
	✘ Request				N/A
	✅ Response	count_i	number		N/A

The fields mentioned above are defined as follows:

Field Name	Example Value	Description
`count_i`	1	Number of times an interaction event occurred with this item
`doc_id`	NMDDV	Product ID or Item ID
`query`	xwearabletech	A query string from the user
`session_id`	91aa66d11af44b6c90ccef44d055cf9a	Id for session in which user generated the signal
`type`	quick_view_click	Type of session the user used to interact with the platform
`user_id`	11506893	ID of user during the session
`timestamp_tdt`	2018-11-20T17:58:57.650Z	Time when signal was generated

Some *signal types*, including *custom signal types*, may include additional fields.

Required Signals for Jobs

A number of Fusion jobs ingest signals. Make sure your signals include the fields required by the jobs you plan to use. The table below shows which jobs require specific fields in your signals.

	query	count_i	type	timstamp_tdt	user_id	doc_id	session_id	fusion_query_id
Created by default								
COLLECTION_NAME_click_signals_aggregation								[*1*]
COLLECTION_NAME_phrase_extraction								
COLLECTION_NAME_spell_correction								
COLLECTION_NAME_synonym_detection								
COLLECTION_NAME_head_tail								
COLLECTION_NAME_user_item_prefs_agg								
COLLECTION_NAME_user_query_history_agg								
Created when recommendations are enabled								
COLLECTION_NAME_bpr_item_recs								
Created when experiments are enabled								

	query	count_i	type	timstamp_tdt	user_id	doc_id	session_id	fusion_query_id
Experiments	✓	✓	✓	✓	✓	✓	✓	✓
Created manually								
Query-to-Query Collaborative Similarity	✓	✓	✓	✓	✓			
Query-to-Query Session-Based Similarity	✓	✓	[*2*]	[*3*]	[*4*]	✓	✓	
Trending Recommender	[*5*]	✓	✓	✓		✓		

1 Required if you are using response signals.

2 Required if you want to weight types differently.

3 Required if you want to use time decay.

4 Required if no `session_id` field is available. Either `user_id` or `session_id` is needed on response signals if doing any click path analysis from signals.

5 Required when identifying trending queries instead of trending items.

The fields mentioned above are defined as follows:

Field Name	Example Value	Description
`count_i`	1	Number of times an interaction event occurred with this item
`doc_id`	NMDDV	Product ID or Item ID
`query`	xwearabletech	A query string from the user
`session_id`	91aa66d11af44b6c90ccef44d055cf9a	Id for session in which user generated the signal
`type`	quick_view_click	Type of session the user used to interact with the platform
`user_id`	11506893	ID of user during the session
`timestamp_tdt`	2018-11-20T17:58:57.650Z	Time when signal was generated

Some *signal types*, including *custom signal types*, may include additional fields.

Boosting

Signals contain data about how users interact with search results. Once your Fusion app has accumulated a sufficient number of *aggregated signals*, they become useful for automatically producing recommendations and boosts for better relevance and higher conversion rates.

The same data used to produce recommendations can also be used for automatic boosting. So although recommendations and boosts are different (as explained below), these topics use "recommender data" to refer to the data that can be used for boosting as well as for recommendations.

Recommendations vs boosts

Recommendations and boosts are two ways of presenting AI-powered estimations about which items are most likely to interest a user:

Recommendations	Boosts
Personalized search results from a special, automatic query, *whether or not the user has performed a query*. Recommendations can be based on a variety of criteria, as in the examples below: • "Based on your purchase history, you might be interested in these items." • "People who viewed this item also viewed these items." • "People who searched for 'ipad' also searched for these items."	Modifications to the *scores* of the items in a query's search results. The set of search results does not change, but their ranking does. For example: • Boost the most popular items • Boost items this user has clicked before • Boost seasonal items Boosts can also be configured manually, using the *Query Workbench* or the *Query Rewriting UI*. You can also perform boosts within a set of recommendations.

These topics explain how to configure and use recommendations and boosts:

- *Getting Started* shows you how to quickly enable the basic feature set and see some results.
- *Recommendation Methods* explains the available approaches to recommendations and boosting, including methods that do not rely on signals.

See also

- *Boost with Signals*

Boost With Signals

The Boost With Signals query pipeline stage performs automatic boosts based on the contents of the aggregated signals collection. For this type of boosting, signals must be enabled but recommendations need not be.

Using the main query and the stage configuration parameters, this stage performs a secondary query to the `COLLECTION_NAME_signals_aggr` collection and returns updated boost weights for the items in the main query's search results. Items that have received more user interaction also receive higher boost weights.

Some of the important configuration parameters are discussed below. For complete details about all configuration parameters for this stage, see *Boost with Signals*.

Configuration overview

The fields below are especially useful to understand when configuring this stage.

Number of Recommendations `numRecommendations`	Sets the `rows` query param in the main query as the maximum number of query results which will be boosted by this pipeline stage.
Number of Signals `numSignals`	Sets the `rows` query param in the query that searches the `COLLECTION_NAME_signals_aggr` collection, so only the specified number of aggregated signals are retrieved and used for boosting. When signals boosting is applied to a query, aggregated signals records are queried from the appropriate `_signals_aggr` collection to find out the popularity or boost weight for documents which have signals. `numSignals` limits the number of records to be queried from a `}}{{_signals_aggr` collection and used to calculate this boost.
Aggregation Type `aggrType`	A filter to retrieve aggregated signals in the `COLLECTION_NAME_signals_aggr` collection per each aggregated signal's `aggr_type_s` field value.
Solr Field to Boost On `boostId`	The document field in the main collection on which to perform boosting. Typically it should use default field, which is `id`. This field corresponds to the **Rollup Field**/`rollupField` field. Together, these two fields act like a `FIELD:VALUE` pair in the query modification for boosting.

Boost Method
 	
`boostingMethod`
	This adds a query parameter to the original query, either "query-param" or "query-parser". The result is `("query-param" or "query-parser") + Boost Param("boost" or `bq`)`, as in the examples below:
	
	"query-param"+"boost", result boost query param
	
	`boost="map(query({!field f='id' v='6239046,13026192}), 0, 0, 1, 27.1705)"`
	
	"query-parser"+"boost", result boost query param
	
	`bp_xxx_bbqx="map(query({!field f='id' v='6239046,13026192}), 0, 0, 1, 27.1705)"`
	
	When `Boost Param` uses `bq`, similar logic applies. When **Boost Param**/`boostingParam` uses "boost", it works with both "query-param" and "query-parser".

Rollup Field `rollupField`	Indicates which aggregated signal document field the boost parameter will use for the final boosting. It works in combination with the **Solr Field to Boost On**/`boostId` field. This should be set to the field in the aggregated signal collection that stores the doc list that is aggregated as one record. By default it's set to `doc_id_s`, because by default this is used in the *click_signal_aggr SQL job*.

Rollup Weight Field `rollupWeightField`	Indicates the final boost weight used to calculate the new score for docs retrieved by the main query. Similar to **Rollup Field**/`rollupField` above, this should be set to the field in the aggregated signal collection that stores the final weight that was calculated. By default it's `weight_d`, because by default this is used in the *click_signal_aggr SQL job*.
Final Boost Weight Expression `weightExpression`	Calculates the final weight using the weight and score retrieved from the `COLLECTION_NAME_signals_aggr` collection. The default value is `math:log(weight_d + 1) + 10 * math:log(score+1)`.

Solr query parameters

These parameters are used in the **Solr Query parameters**/`queryParams` field for retrieving signal aggregation docs from the `COLLECTION_NAME_signals_aggr` collection. These Solr query params will affect which aggregated signals are used for producing the boosting parameter on the main query.

`qf=query_t`	Defines which field to query. In the default case, the query searches on the `query_t` field of aggregated signal docs.
`pf=query_t^50`	Boosts docs within the set of retrieved docs using phrase matching.
`pf2=query_t^20`	`pf2` is similar to `pf`; the difference is that `pf2` works on bigram phrases.
`pf3=query_t^10`	`pf3` is similar to `pf`; the difference is that `pf3` works on trigram phrases.

FAQs

If there is `fq` in the main query, how is it matched with the correct aggregated signal?
In this case, you need to use the `lw.rec.fq` query parameter in the main query. `lw.rec.fq` can be parsed by the Boost with Signals stage, and therefore the filters specified in it can be added to the Solr query that is retrieving the aggregated signals.

For example, if we have filter query param `fq=format:CD&fq=name:Latin`, this needs to be translated into `lw.rec.fq=filters_s:"format:cd $ name:latin"`. Values must be lowercase. The final main query should be:

```
http://FUSION_HOST:FUSION_PORT/api/apps/demo_app/query-pipelines/demo_app/collections/demo_app/select?echoParams=all&wt=json&json.nl=arrarr&sort&start=0&q=apple&debug=true&rows=10&lw.rec.fq=filters_s:"format:cd $ name:latin"
```

Now the Boost with Signals stage will only retrieve aggregated signals that have the same filter query.

> If there are multiple `fq` values (for example, `format:cd` and `name:latin`), they are ordered alphabetically as strings and joined with " $ " (a $ with a space on each side). In the example, `"format:cd $ name:latin"`.

What if my aggregated signals are in a different collection?

You can point the Boost with Signals stage to a different signal collection by adding a `collection` parameter in the `Solr Query Parameters` section.

Recommendations

Signals contain data about how users interact with search results. Once your Fusion app has accumulated a sufficient number of *aggregated signals*, they become useful for automatically producing recommendations and boosts for better relevance and higher conversion rates.

The same data used to produce recommendations can also be used for automatic boosting. So although recommendations and boosts are different (as explained below), these topics use "recommender data" to refer to the data that can be used for boosting as well as for recommendations.

Recommendations vs boosts

Recommendations and boosts are two ways of presenting AI-powered estimations about which items are most likely to interest a user:

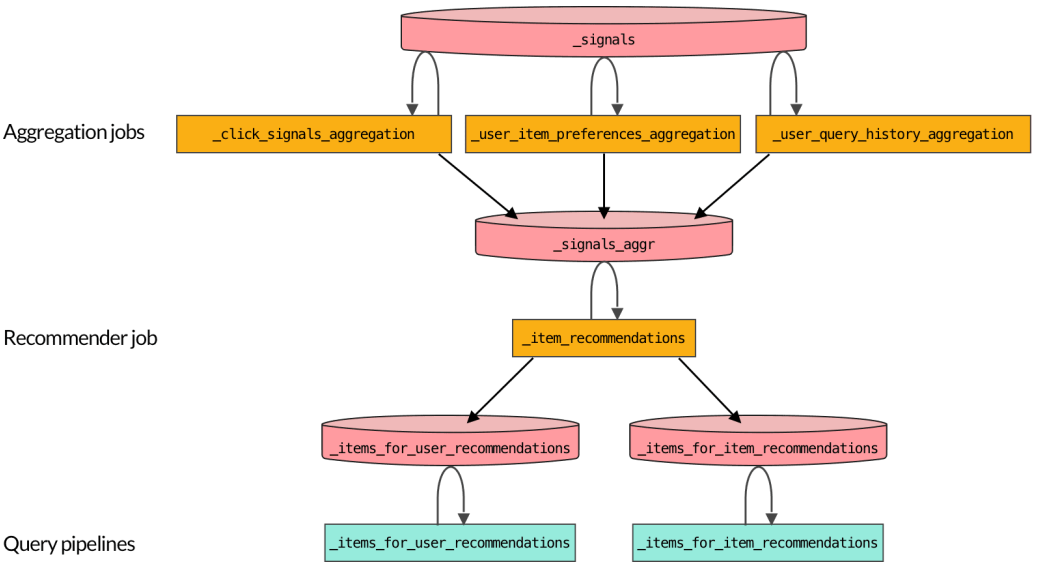
Recommendations	Boosts
Personalized search results from a special, automatic query, *whether or not the user has performed a query*. Recommendations can be based on a variety of criteria, as in the examples below: • "Based on your purchase history, you might be interested in these items." • "People who viewed this item also viewed these items." • "People who searched for 'ipad' also searched for these items."	Modifications to the *scores* of the items in a query's search results. The set of search results does not change, but their ranking does. For example: • Boost the most popular items • Boost items this user has clicked before • Boost seasonal items Boosts can also be configured manually, using the *Query Workbench* or the *Query Rewriting UI*. You can also perform boosts within a set of recommendations.

These topics explain how to configure and use recommendations and boosts:

- *Getting Started* shows you how to quickly enable the basic feature set and see some results.
- *Recommendation Methods* explains the available approaches to recommendations and boosting, including methods that do not rely on signals.

Recommendations data flow

The diagram below shows the flow of data between the *default objects* created when you *enable recommendations*. You can create additional object for different *recommendation methods*.



1. *Signals data* is aggregated into a format that can be consumed by recommender jobs.
   Your search application should be sending *well-formed signals data* to Fusion whenever there is user activity on your site.
   By default, the `COLLECTION_NAME_click_signals_aggregation` job runs every 15 minutes while the `_user_item_preferences_aggregation` and `_user_query_history_aggregation` jobs run once per day.

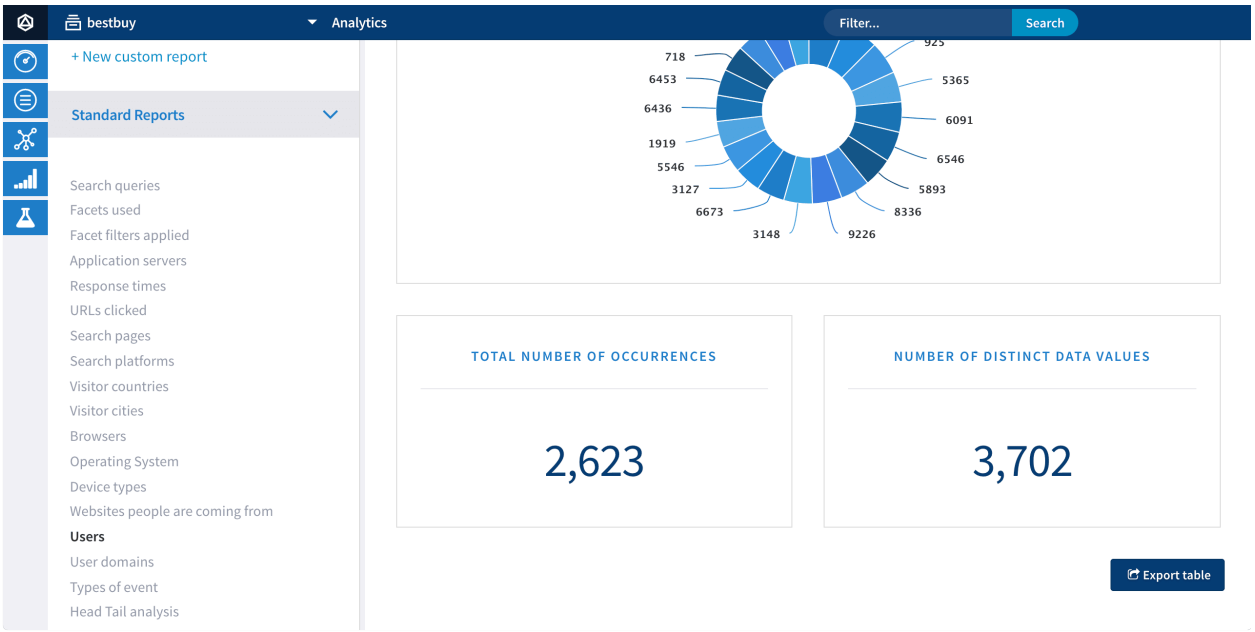
You can use signals data for boosting, without enabling recommendations, using the *Boost with Signals* query pipeline stage. The *default query pipeline* includes this stage.

2. *Recommender data* is produced when recommender jobs analyze aggregated signals.
   The default `COLLECTION_NAME_item_recommendations` job must be manually *scheduled* or run on demand. Make sure that the `COLLECTION_NAME_user_item_preferences_aggregation` job runs first.
3. Query pipelines retrieve recommender data to produce recommendations and boosts.
   When recommendations are enabled, Fusion creates two query pipelines for recommendations:
   `COLLECTION_NAME_items_for_user_recommendations`
   `COLLECTION_NAME_items_for_item_recommendations`

Recommendations storage requirements

Collaborative recommendations are derived from data generated by Fusion *jobs* and stored in special collections. You can estimate the required storage for this data if you know the following:

- the number of unique users in the signals collection (`num_users`)
  To find the number of unique values of the `user_id` field, navigate to **Analytics** > **App Insights** > **Analytics** > **Users**. The **Number of Distinct Data Values** panel displays the number of unique users:



  If no value is displayed here or the value is not accurate, see *Important fields for signals* to verify that your signals data is well-formed.
- the number of unique items in the main collection (`num_items`)
  Navigate to **Collections** > **Collections Manager** to see the total number of documents in your main collection.
- the number of recommendations to be computed per user (`num_recs_per_user`)
  This is a configuration parameter in the recommender job, with a default value of 10.
- the number of item similarities to be computed per item (`num_item_sims`)
  This is a configuration parameter in the *recommender job*, with a default value of 10.

Every time the recommender job runs, this many new recommender documents are created:

```
num_users * num_recs_per_user + num_items * num_item_sims
```

Each of these documents is about 300 bytes. This is the minimum size; they can be larger if you configure the recommender job to pull item metadata fields from the main collection for inclusion in recommender documents. These fields are useful when displaying recommendations in your front-end search application.

	In your recommender jobs, verify that **Delete Old Recommendations** is selected so that only the latest recommender data is stored.

Additional resources

- View our training on Recommendations *(https://academy.lucidworks.com/recommendations-course)*.

Getting Started with Recommendations and Boosting

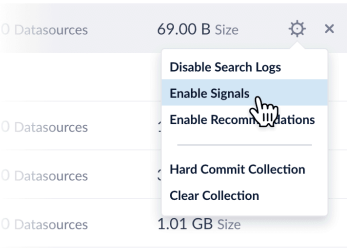
Signals provide the data that Fusion uses to generate *collaborative recommendations*. The simplest way to get started is to enable signals and recommendations in one of your primary collections.

Once you do this, Fusion automatically creates a set of *default objects* and begins creating and updating collections of recommendations on a regular schedule.

	Content-based recommendations can be used without enabling signals or recommendations, but they require manual configuration.

Enabling signals

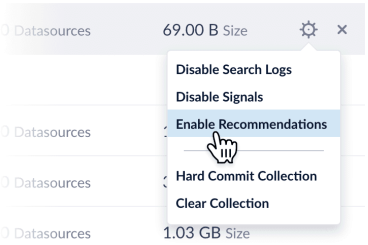
Signals are enabled by default for new collections when you have a Fusion license installed. You can enable or disable signals for any collection at **Collections** > **Collections Manager**.



Enabling signals automatically creates a set of *aggregation jobs* which create the input data for recommendations. See *Signals and Aggregations* for complete details.

Enabling recommendations

Recommendations are not enabled by default; you can do this at **Collections** > **Collections Manager**.



When you enable recommendations, this automatically enables the *items-for-user* and *items-for-item* recommendation methods. To use additional *recommendation methods*, you must configure them separately.

Default objects for recommendations

When recommendations are enabled, Fusion automatically creates a default set of collections, jobs, schedules, and query pipelines that provide basic functionality for recommendations.

You can tune the default jobs and pipelines as needed to refine the results, or create new ones, then configure your search application to request recommendations from the query pipelines.

See also the default objects created when you *enable signals*. These must already exist when you enable recommendations.

Collections

- `COLLECTION_NAME_items_for_item_recommendations`
  Collection to hold generated item-item similarities (by default 10 per item). No `user_id_s` data is present. A Recommend Items for Item query pipeline stage can use the similarities to return item recommendations. For example, a query in which `doc_id_s = docA` would return an ordered list of other `doc_id_s` values for documents that are similar to document `docA`, along with the similarities. For example: `[("docB", 0.83), ("docC", 0.55), ("docD", 0.43), …, ("docK", 0.22)]`.
- `COLLECTION_NAME_items_for_user_recommendations`
  Collection to hold recommended items for a user. By default the job creates 10 recommendations per user.
- `COLLECTION_NAME_recs_aggr`
  Collection to hold the output of recommender aggregation jobs.
- `COLLECTION_NAME_bpr_user_recs` and `COLLECTION_NAME_bpr_item_recs`

Collections for Bayesian Personalized Ranking (BPR) recs, user-item, and item-item (respectively).

- `COLLECTION_NAME_content_recs`
  Collection for content-based recommender recs.
- `COLLECTION_NAME_recommender_models`
  Collection that stores ALS models.
- `COLLECTION_NAME_queries_query_recs`
  Collection for output of the Query-Query Similarity Job.

Jobs and schedules

Enabling recommendations creates the new jobs below, each using a different method to produce items-for-item and items-for-user recommendations. These jobs consume the output of the *signals aggregation jobs*.

	BPR Recommender	***Content-Based Recommender***	***Query-to-Query Session-Based Similarity***	**Default job name**
`COLLECTION_NAME_item_recommendations`	`COLLECTION_NAME_bpr_item_recs`	`COLLECTION_NAME_content_recs`	`COLLECTION_NAME_query_recs`	Default input collection
`COLLECTION_NAME_signals_aggr`	`COLLECTION_NAME_signals_aggr`	`COLLECTION_NAME`	`COLLECTION_NAME_signals`	Default output collections
`COLLECTION_NAME_items_for_user_recommendations` `COLLECTION_NAME_items_for_item_recommendations`	`COLLECTION_NAME_bpr_user_recs` `COLLECTION_NAME_bpr_item_recs`	`COLLECTION_NAME_content_recs`	`COLLECTION_NAME_queries_query_recs`	Recommendation type

None of the jobs are scheduled by default; you must manually run or schedule them.

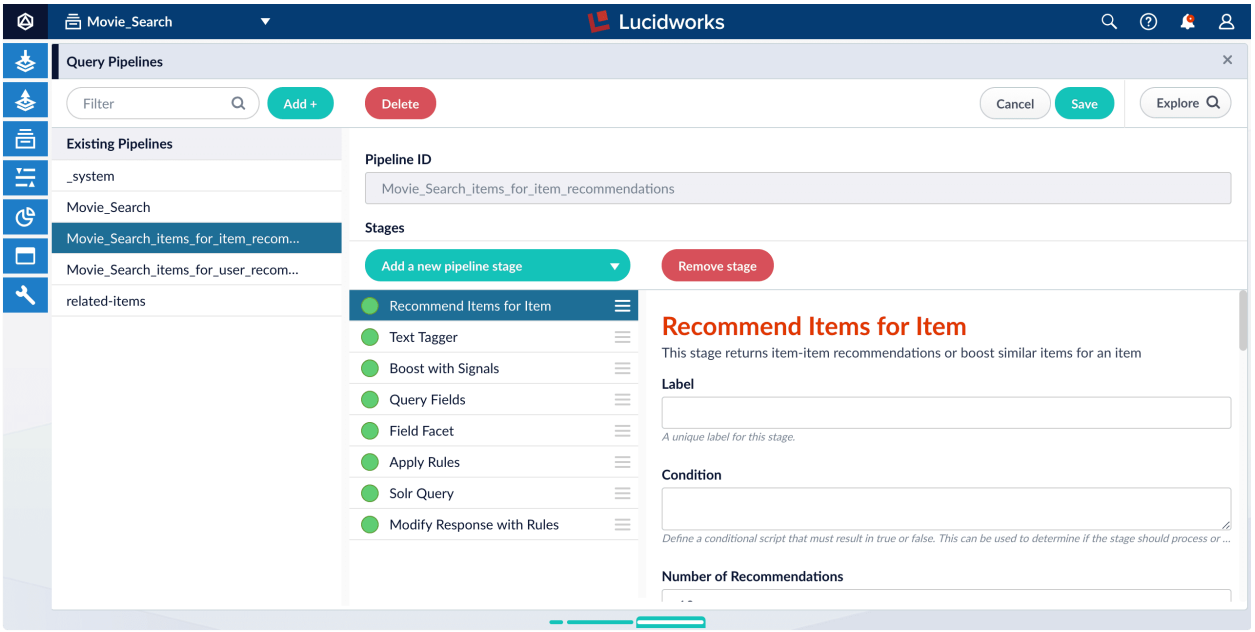
NOTE:

- The `COLLECTION_NAME_user_item_prefs_agg` job provides input data for the ALS and BPR jobs and must run before them. See *SQL Aggregations* for details.
- The `COLLECTION_NAME_user_query_history_agg` job generates aggregations for user-item preferences from signal information.

Fusion does *not* automatically create a *Query-to-Query Session-Based Similarity job*, which is needed for certain *recommender types*.

Query pipelines

- `COLLECTION_NAME_items_for_user_recommendations`
  Query pipeline to generate recommendations of items for a user.
- `COLLECTION_NAME_items_for_item_recommendations`
  Query pipeline to generate recommendations of items similar to an item.



- `COLLECTION_NAME_queries_query_recs`
  Query pipeline to generate *queries-for-query recommendations* using a model created by the *Query-to-Query Session-Based Similarity* job.

Methods for Recommendations and Boosting

This topic provides an overview of different methods for generating recommendations and boosts. "Content-based" and "collaborative" recommendations are two broad categories. More specific recommendation methods are also explained.

Content-based vs collaborative recommendations

Recommendation methods can be divided into two broad categories: collaborative recommendations and content-based recommendations.

Collaborative recommendations always require signals. Some also require a job that pre-computes a matrix of values that can be queried. When these values are used for boosting, then certain query pipeline stages are needed as well.

Method	Requires aggregated signals	Required job	Boosting stage
Collaborative recommendations These methods analyze how users have interacted with documents.			
Items-for-item	✓	*BPR Recommender*	`APPName_item_item_rec_pipelines_bpr.json`
Items-for-user	✓	*BPR Recommender*	*Recommend Items for User*
Items-for-query	✓	*Query-to-Query Session-Based Similarity jobs*	
Queries-for-query	✓	*Query-to-Query Session-Based Similarity jobs*	
Boost with signals	✓		*Boost with Signals*
Trending items or queries	✓	*Trending Recommender*	
Users-for-item recommendations	✓	*BPR Recommender*	
Content-based recommendations These methods analyze the content of documents or queries.			
Items-for-item		*Content-Based Recommender*	`APPName_item_item_rec_pipelines_content.json`
Boost documents			*Boost Documents*
Parameterized boosting			*Parameterized Boosting*
More Like This			*Solr MoreLikeThis*
Business rules			*Apply Rules*

Fusion recommendation and boosting methods

This section provides an overview of each method, with links to more detailed topics.

Collaborative recommendations	Content-based recommendations
• *Items-for-item* • *Items-for-user* • *Items-for-query* • *Queries-for-query* • *Boost with signals* • *Trending items or queries* • *Users-for-item*	• *Items-for-item* • *Boost Documents* • *Parameterized Boosting* • *More Like This* • *Business rules*

When you *enable recommendations*, this automatically enables the items-for-user and items-for-item recommendation methods. To use additional recommendation methods, you must configure them separately.

Items-for-item recommendations

Items-for-item recommendations present items that are similar to a specified item. For example, when the user is viewing a BMX bicycle, Fusion can recommend other BMX bicycles. Similarity can be based on different criteria, such as click patterns, people who bought this also bought that, percentage match of document tags, and so on.

See *Items-for-Item Recommendations* for details.

Items-for-user recommendations

Items-for-user recommendations use the *Recommend Items for User* query stage to present items that are similar to ones in which the user has previously shown interest, based on the user's search history, browsing history, purchase history, and so on. This is one type of *collaborative recommendation*.

See *Items-for-User Recommendations* for details.

Items-for-query recommendations

The Items for Query recommender is the primary algorithm that supports recommendations on the search results page.

Items-for-query recommendations are based on how other users interacted with search results from the same query. This is one type of *collaborative recommendation*.

For example, if users have searched for "titanic" in the past and many of them clicked on the search result for the DVD of the movie *Titanic* (as opposed to books or memorabilia), then this method boosts the DVD item on subsequent searches for "titanic".

See *Items-for-Query Recommendations* for details.

Queries-for-query recommendations

Queries-for-query recommendations are queries performed by other users who also performed the current query. For example, when a user searches for "madonna", your app may also suggest searches for "evita", "a league of their own", "lady gaga", and so on.

See *Queries for Query* for details.

Boost with signals

The Boost With Signals query pipeline stage performs automatic boosts based on the contents of the aggregated signals collection. For this type of boosting, signals must be enabled but recommendations need not be.

See *Boost With Signals* for details.

Trending items or queries

The *Trending Recommender job* analyzes signals to measure customer engagement over time. Use this job to identify spikes in popularity for specific items or queries, then display those items to your users or analyze the trends for business purposes. You can configure any time window, such as daily, weekly, or monthly.

See *Trending items or queries* for details.

Users-for-item recommendations

Users-for-item recommendations retrieve the set of users who have interacted with an item, weighted by the number of interactions. This can be useful for community-driven use cases or for marketing campaigns. For example, on your organization's intranet, clicking a search result could display the item plus a list of colleagues who have interacted with the same item.

See *Users-for-Item Recommendations* for details.

Boost documents

The Boost Documents query pipeline stage adds boosting parameters to matched documents based on user-defined rules. Boosts are defined with a term value to boost and the boost factor to add. The boosting parameters are added to the `bq` Solr query parameter.

See *Boost Documents* for details.

Parameterized boosting

The Parameterized Boosting query pipeline stage reads the `boostValues` (in `List<DocumentResult>` format) from the context variable (added by a *Rollup Aggregation stage* or a *JavaScript stage*), and adds boosts to the main query using `bq` or `boost` based on the stage configuration. The weights for the boost values can also be scaled.

See *Parameterized Boosting* for details.

More Like This

This stage uses the content of the current document to query for similar documents, using *Solr's MoreLikeThis component*.

This stage provides content-based recommendations. For collaborative recommendations, use the *Recommend Items for Item stage*.

See *Solr MoreLikeThis* for details.

Business rules

Business rules are manually-created formulas for rewriting queries. This is the most versatile strategy for creating custom query rewrites. It supports a variety of conditions and actions to address a wide range of use cases. When you need a very specific query rewrite, this is the best strategy.

Business rules are applied in the *Apply Rules stage* of the query pipeline.

See *Business Rules* to learn how to create, edit, and publish business rules. See *Query Rewriting* for additional query rewriting methods.

BPR algorithm

The Bayesian Personalized Ranking (BPR) algorithm is used to configure items-for-item, items-for-user, or user-for-item recommendations. The BPR job generates effective results in a short run time. See the *BPR Recommender* for more information.

Items-For-Item Recommendations

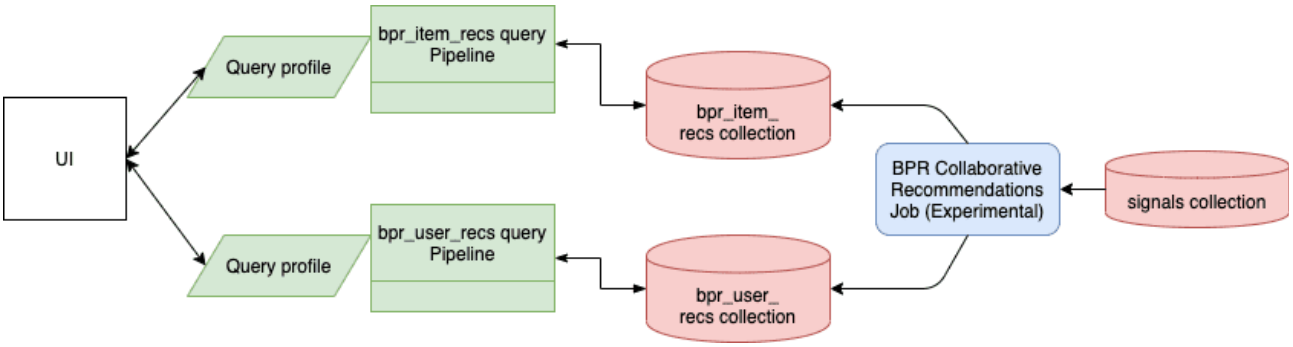
Items-for-item recommendations present items that are similar to a specified item. For example, when the user is viewing a BMX bicycle, Fusion can recommend other BMX bicycles. Similarity can be based on different criteria, such as click patterns, people who bought this also bought that, percentage match of document tags, and so on.

Collaborative items-for-item recommendations

If you have enabled signals and recommendations for a collection, then the default `COLLECTION_NAME_bpr_item_recs`, a *BPR Recommender* job is already created and configured to produce items-for-item recommendations (as well as *items-for-user recommendations*).

This is the recommended job type, and produces better results with a short runtime. *Download the* `APPName_item_item_rec_pipelines_bpr.json` file and *import it* to create the query pipeline that consumes this job's output. See *Fetch Items-for-Item Recommendations (Collaborative/BPR Method)* for details.

BPR collaborative recommendations dataflow

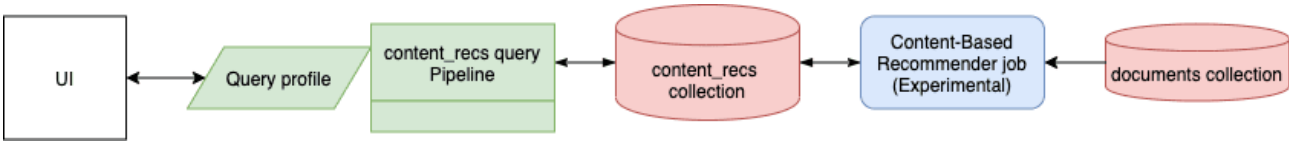


	If you want to use different parameters for items-for-item recommendations and *items-for-user recommendations*, simply create separate jobs for each, where one job configuration includes an output collection for items-for-item recommendations only and the other includes an output collection for items-for-user recommendations only.

Content-based items-for-item recommendations

The *Content-Based Recommender* job computes item similarities based on their content, such as product descriptions. This is a useful option when you do not have enough signals for accurate results using the collaborative method described above. *Download the* `APPName_item_item_rec_pipelines_content.json` file and *import it* to create the query pipeline that consumes this job's output.

Content-based recommendations dataflow



Once you have run the recommender job, see *Fetch Content-Based-Items-for-Item Recommendations*.

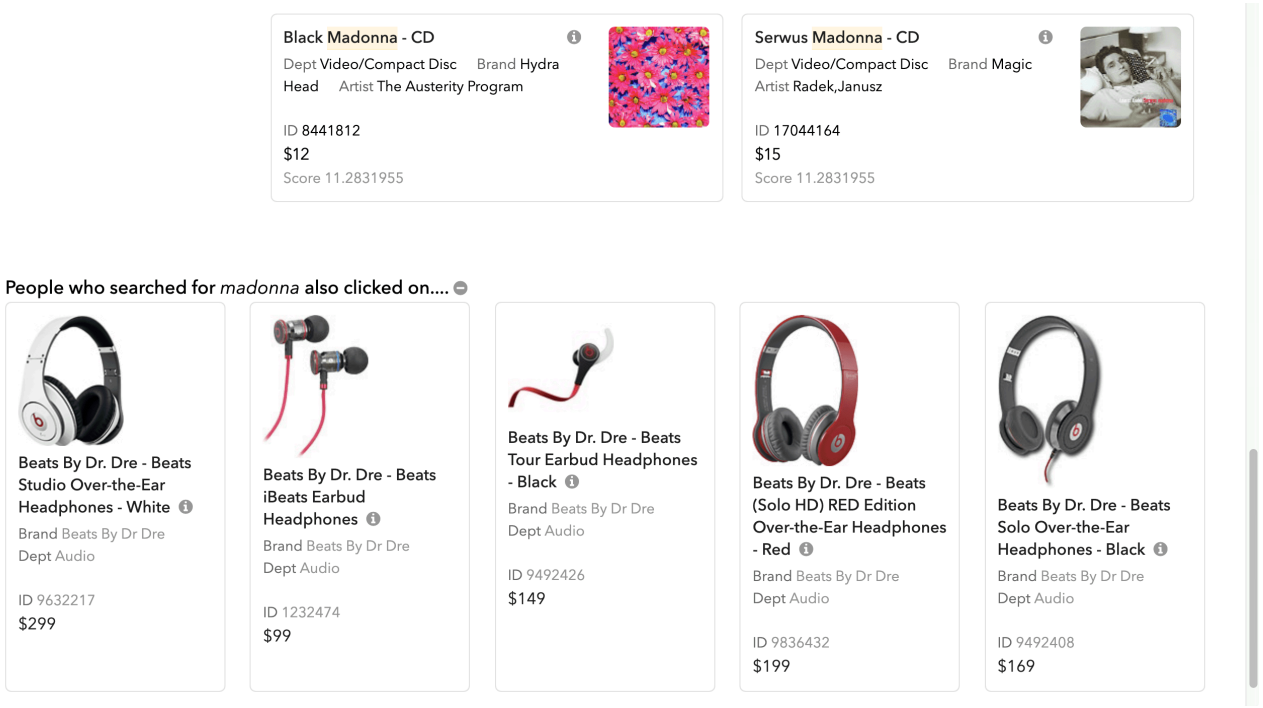
Items-For-Query Recommendations

The Items for Query recommender is the primary algorithm that supports recommendations on the search results page.

Items-for-query recommendations are based on how other users interacted with search results from the same query. This is one type of *collaborative recommendation*.

For example, if users have searched for "titanic" in the past and many of them clicked on the search result for the DVD of the movie *Titanic* (as opposed to books or memorabilia), then this method boosts the DVD item on subsequent searches for "titanic".

Below is an example of items-for-query recommendations based on a search for "madonna":



To use this type of boosting, signals must be enabled but recommendations need not be.

Job configuration

The items-for-query recommendations are implemented as a *Query-to-Query Session-Based Similarity* job.

Query pipeline configuration

The *Boost with Signals* query stage is part of the default query pipeline, so usually you do not need to add it. It takes its input from the aggregated signals collection (`COLLECTION_NAME_signals_aggr`) at query time.

Items-For-User Recommendations

Items-for-user recommendations use the *Recommend Items for User* query stage to present items that are similar to ones in which the user has previously shown interest, based on the user's search history, browsing history, purchase history, and so on. This is one type of *collaborative recommendation*.

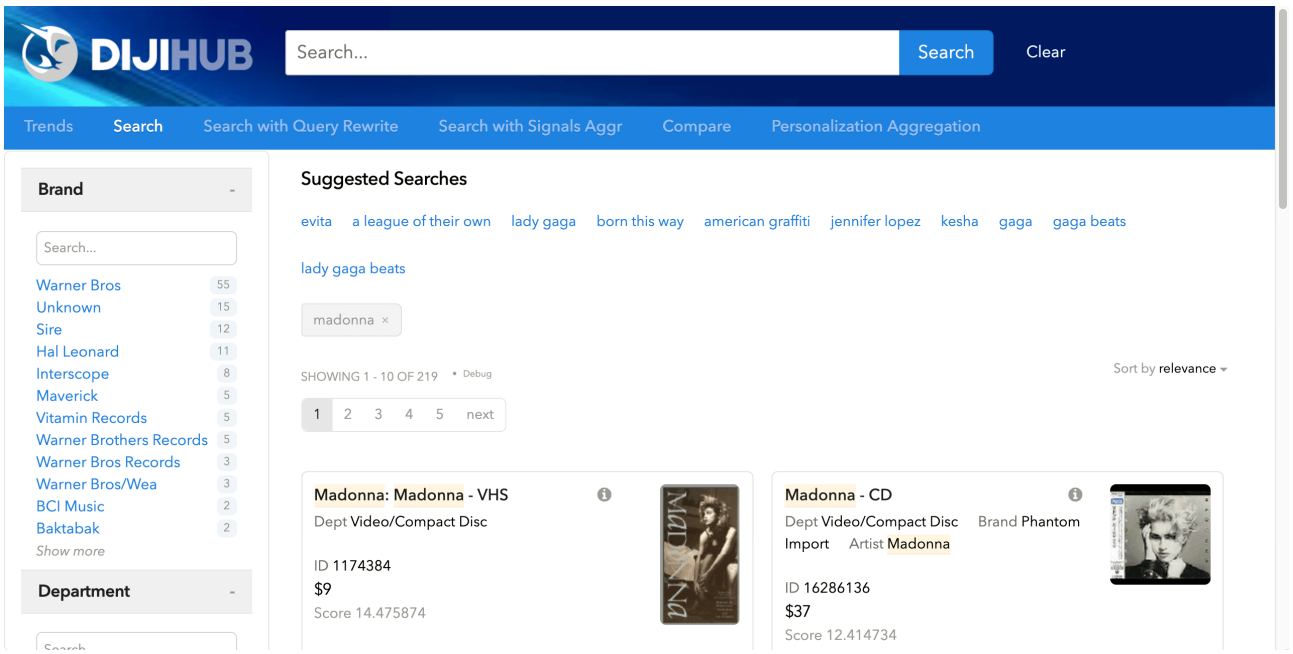
If you have enabled signals and recommendations for a collection, then the default `COLLECTION_NAME_bpr_item_recs`, a *BPR Recommender* job is already created and configured to produce items-for-user recommendations.

This is the recommended job type, and produces better results with a short runtime. *Download the* `APPName_item_item_rec_pipelines_bpr.json` file and *import it* to create the query pipeline that consumes this job's output. See *Fetch Items-for-User Recommendations (Collaborative/BPR Method)* for details.

Queries-for-Query Recommendations

Queries-for-query recommendations are queries performed by other users who also performed the current query. For example, when a user searches for "madonna", your app may also suggest searches for "evita", "a league of their own", "lady gaga", and so on.

The "Suggested Searches" shown below are one example of queries-for-query recommendations:



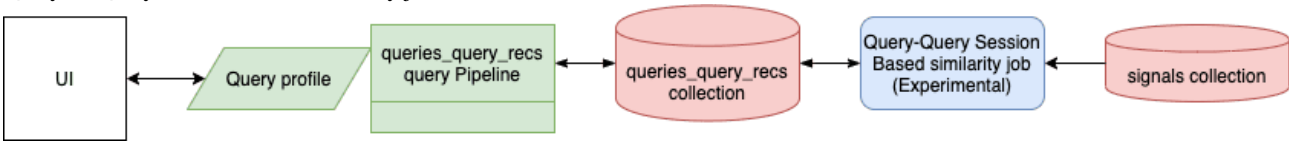
To enable queries-for-query recommendations, use the *Query-to-Query Session-Based Similarity* job, which is session-based unlike the *Synonym Detection job*.

The Query-to-Query Similarity recommender job was deprecated in Fusion 5.2.

The job generates two sets of recommendations based on the two approaches described below, then merges and de-duplicates them to present unique query-recommender pairs.

Similar queries based on documents clicked	Similar queries based on co-occurrence in sessions
Queries are considered for recommendation if two queries have similar sets of document IDs clicked according to the signals data. This is directly implemented from the similar queries portion of the *Synonym Detection job*. This approach can work on both raw and aggregated signals.	Queries are considered for recommendation if two queries have co-occurred in the same session based on the assumption that users search for similar items in a single search session (this may or may not hold true depending on the data). This approach, based on session/user IDs, needs raw signals to work.

Query-to-Query Session-Based Similarity job dataflow



A default Query-to-Query Session-Based Similarity job (`COLLECTION_NAME_query_recs`) and a dedicated *collection* and *pipeline* are created when you *enable recommendations for a collection*.

At a minimum, you must configure these:

- an ID for this job
- the input collection containing the signals data, usually `COLLECTION_NAME_signals`
- the data format, usually `solr`
- the query field name, usually `query_s`
- the document ID field name, usually `doc_id_s`
- optionally, the user session ID or user ID field name

  If this field is not specified, then the job generates click-based recommendations only, without session-based recommendations.

Data tips

- Running the job on other types of data than signals is not recommended and may yield unexpected results.
- To get about 90% query coverage with the query pipeline, we recommend a raw signals dataset of about ~170k unique queries. More signals will generally improve coverage.
- On a raw signal dataset of about 3 million records, the job finishes execution in about 7-8 minutes on two executor pods with one CPU and about 3G of memory each. Your performance may vary depending on your configuration.

Boosting recommendations

Generally if a query and recommendation has some token overlap, then they're closely related and we want to highlight these. Therefore, query-recommendation pair similarity scores can be boosted based on token overlap. This overlap is calculated in terms of the number or fractions of tokens that overlap.

For example, consider the pair ("a red polo shirt", "red polo"). If the minimum match parameter is set to 1, then there should be 1 token in common. For this example there is 1 token in common and therefore it is boosted. If it is set to 0.5, then at least half of the tokens from the shorter string (in terms of space separated tokens) should match. Here, the shorter string is "red polo" which is 2 tokens long. Therefore, to satisfy the boosting requirement, at least 1 token should match.

Tuning tips

These tuning tips pertain to the advanced Model Tuning Parameters:

- **Special characters to be filtered out.** Special characters can cause problems with matching queries and are therefore removed in the job.
  Only the characters are removed, not the queries, so a query like `ps3$` becomes `ps3`.
- **Query similarity threshold.** This is for use by the similar queries portion of the job and is the same as that used in the Synonym and Similar Queries Detection job.
- **Boost on token overlap.** This enables or disables boosting of query recommendation pairs where all or some tokens match. How much match is required to boost can be configured using the next parameter.
  For example, if this is enabled, then a query-recommendation pair like `(playstation 3, playstation console)` is boosted with a similarity score of 1, provided the minimum match is set to 1 token or 0.5.
- **Minimum match for token overlap.** Similar to the `mm` param in Solr, this defines the number/fraction of tokens that should overlap if boosting is enabled. Queries and recommendations are split by " " (space) and each part is considered a token. If using a less-than sign (<), it must be escaped using a backslash.
  The value can be an integer, such as 1, in which case that many tokens should match. So in the previous example, pair is boosted because the term "playstation" is common to both and the `mm` value is set to 1.

  The value can also be a fraction, in which case that fraction of the shorter member of the query and recommendation pair should match. For example, if the value is set to 0.5 and query is 4 tokens long and recommendation is 6 tokens long then there should be at least 2 common tokens between query and recommendation.

  Here the stopwords specified in the *list of stopwords* are ignored while calculating the overlap.
- **Query clicks threshold.** The minimum number of clicked documents needed for comparing queries.
- **Minimum query-recommendation pair occurrence count.** Minimum limit for the number of times a query-recommendation pair needs to be generated to make it to the final similar query recommendation list. Default is set to 2. Higher value will improve quality but reduce coverage.

The similar queries collection

The following fields are stored in the `COLLECTION_NAME_queries_query_recs` collection:

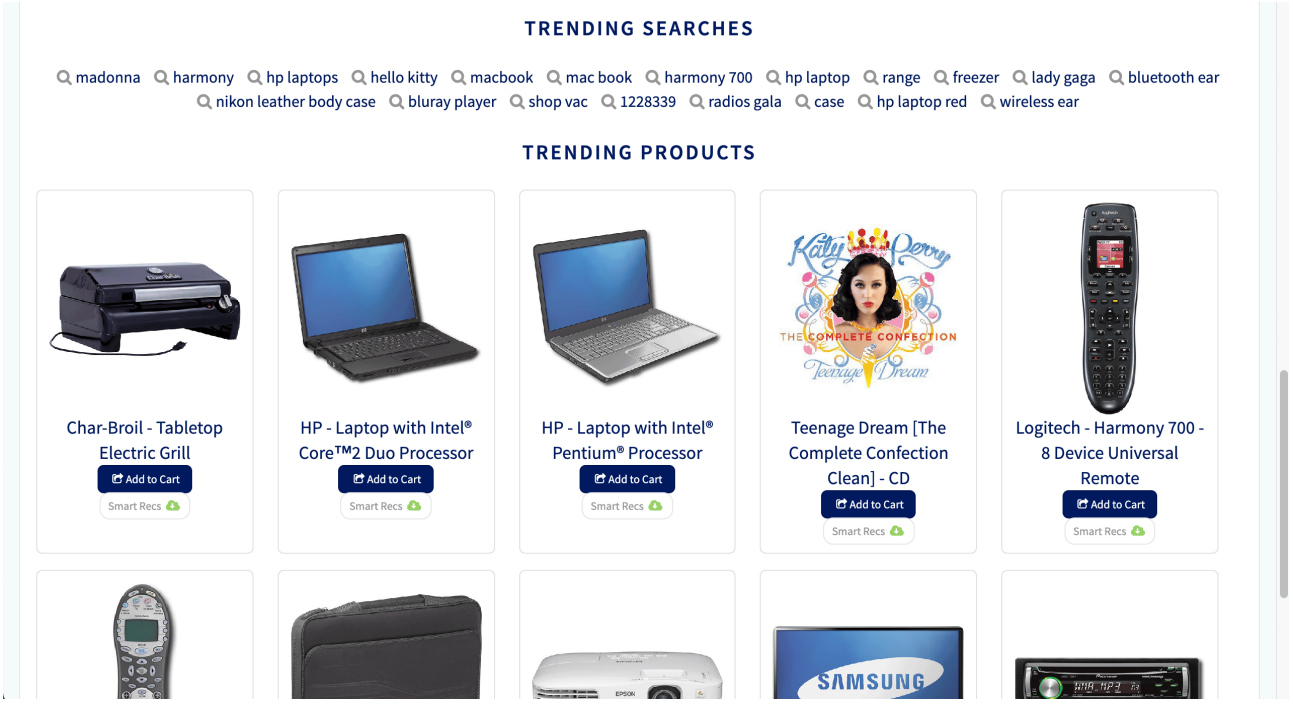
- `query_t`
- `recommendation_t`
- `similarity_d`, the similarity score
- `source_s`, the approach that generated this pair, one of the following: `SessionBased` or `ClickedDocumentBased`
- `query_count_l`, the number of times the query occurred in signals
- `recommendation_count_l`, the number of times recommendations occurred in signals
- `pair_count_l`, the number of instances of the pair generated in the final recommendations using either of the approaches
- `type_s`, always set to `similar_queries`

The query pipeline

When you enable recommendations, a default query pipeline, `COLLECTION_NAME_queries_query_recs`. is created.

Trending Items and Queries

The *Trending Recommender job* analyzes signals to measure customer engagement over time. Use this job to identify spikes in popularity for specific items or queries, then display those items to your users or analyze the trends for business purposes. You can configure any time window, such as daily, weekly, or monthly.



For detailed steps to configure this job, see *Identify Trending Documents or Products*.

Users-for-Item Recommendations

Users-for-item recommendations retrieve the set of users who have interacted with an item, weighted by the number of interactions. This can be useful for community-driven use cases or for marketing campaigns. For example, on your organization's intranet, clicking a search result could display the item plus a list of colleagues who have interacted with the same item.

This is one type of *collaborative recommendation*.

Synonyms

Synonyms are interchangeable terms that can be used to expand a query to retrieve more or better search results. For example, "TV" and "television" can be considered synonyms; when a user enters one of these terms, Fusion can also look for results containing the other term.

Synonyms Files

Synonyms

Synonyms are:

- Words that mean the same thing, within the context where they are used.
- Used in searches. And synonym expansion allows Fusion to return results that match the meaning of the query terms, but not the words themselves.
- Important for mapping query terms such as:
  - Acronyms to their names
  - Jargon to public terms
  - Misspellings to correct spellings
  - Old to new personal or corporate names
  - Bridging the gap between the user vocabulary and terms in the original text

Fusion uses the Solr `synonyms.txt` and Solr collections, which are managed by Fusion.

Solr itself manages a set of resources *(https://cwiki.apache.org/confluence/display/solr/Managed+Resources)* to apply synonym expansion, with configuration through the Fusion API and the Fusion UI. However, Fusion synonyms are not interchangeable with Solr synonyms files.

If you have a Fusion license, see also the *Synonym Detection Jobs* which automatically detects synonyms to use in *query rewriting*.

See *Use Synonym Detection* for more information.

Synonym types

There are three kinds of search synonyms, depending on the requirements of the search for each specific term.

Replacement synonyms

Replacements are used to change the query, to replace it with a more standard term or terms. For example:

```
lucid => lucidworks
```

In this case, "lucid" by itself not an approved term, so there should be no instances where the company name is a partial word.

One-way expansion synonyms

One-way expansions expand original terms with more standard terms while retaining the original term; they do not do the opposite, expand standard terms to the original non-standard terms:

```
monitor => monitor, display
```

In this case, "display" is the standard term, but "monitor" is used in some older user-generated content.

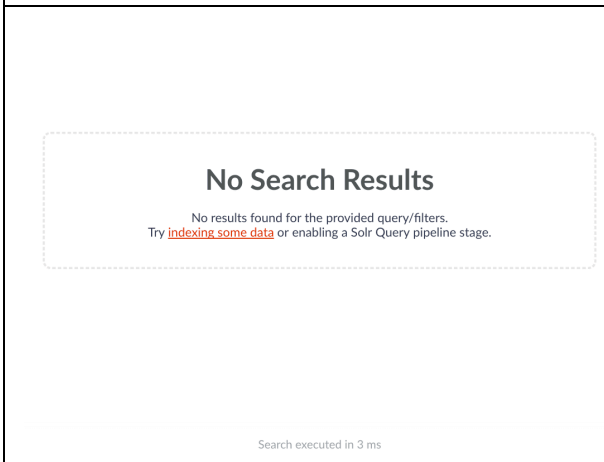
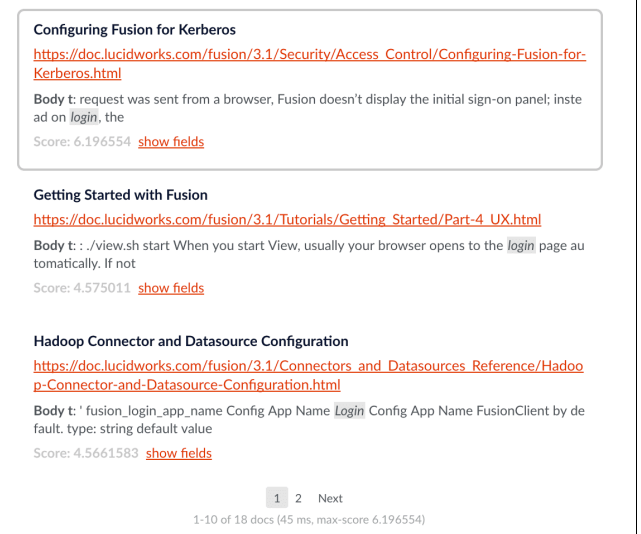
Multi-way expansion synonyms

Where each term is considered equally standard, multiway synonyms expand the query so any items with any of the terms is retrieved:

```
login,logon,signin,signon
```

This example shows terms that are used interchangeably by authors. For this search engine, there is no need to distinguish among them, and considerable value in increasing recall to find all items discussing this topic. However, other content stores may use them differently. Note that "logging" and "signing" have specific meanings in many contexts, so they might not be candidates for synonyms.

Example of synonym expansion:

Results before synonyms	Results after synonyms
	

Viewing the query using the debug=true parameter shows how it is expanded:

```
"querystring": "logon", "parsedquery": "(+DisjunctionMaxQuery((Synonym(_text_:login _text_:logon _text_:signin _text_:signon))))/no_coord", "parsedquery_toString": "+(Synonym(_text_:login _text_:logon _text_:signin _text_:signon))",
```

Multiword synonyms

Lucene/Solr supports multiword synonyms in version 6.6 and later, and Fusion in version 3.1 and later. There are significant technical complexities in performing graphed phrased expansion that had to be overcome.

To enable multiword synonyms in Fusion, create an Additional Parameter stage for disabling the split on whitespace tokenization process (which applies to synonyms only):

```
sow=false
```

Using EDismax, this allows the new Solr SynonymGraphFilter to create the graph representations of token streams containing overlapping synonyms of varying word counts, and expand the queries with additional terms. If using a less-than sign (<) with EDismax, it must be escaped using a backslash.

Examples:

```
appstudio => app studio signup =>signup,sign up login,log in,logon,log on,signin,sign in,signon,sign on
```

Multiword synonyms work just like single-word synonyms, expanding the parsed query with additional query terms. For Solr details, see: Multi-Word Synonyms: Solr Adds Query-Time Support *(https://lucidworks.com/post/multi-word-synonyms-solr-adds-query-time-support//)*.

Business Rules

Business rules are manually-created formulas for rewriting queries. This is the most versatile strategy for creating custom query rewrites. It supports a variety of conditions and actions to address a wide range of use cases. When you need a very specific query rewrite, this is the best strategy.

Business rules are applied in the *Apply Rules stage* of the query pipeline.

To learn how to use business rules in the *Rules Editor*, see *Use Business Rules in the Rules Editor*.

Configuration options

General

General configurations are used to characterize a rule, including its name, description, group, and more. They are also used to set a rule's precedence, or priority over other rules.

For complete details, see *General Configurations*.

Condition

Conditions, or triggers, define the environment necessary for the rule action to fire. Any combination of conditions (date, field value, query, and query profile) can be defined for a rule.

For complete details, see *Conditions*.

Action

Business rule actions define what actions are taken when a rule fires.

For complete details, see *Actions*.

Custom Rule Actions

Custom rule actions allow you to define a unique action to take when certain conditions are met. They involve three primary components: a query pipeline, a custom rule type, and a rule that is triggered by conditions defined by the user.

For complete details, see *Custom Rule Actions*. To learn use custom rule actions with the API, see *Create Custom Rule Actions with the API*.

Query Elevation Component

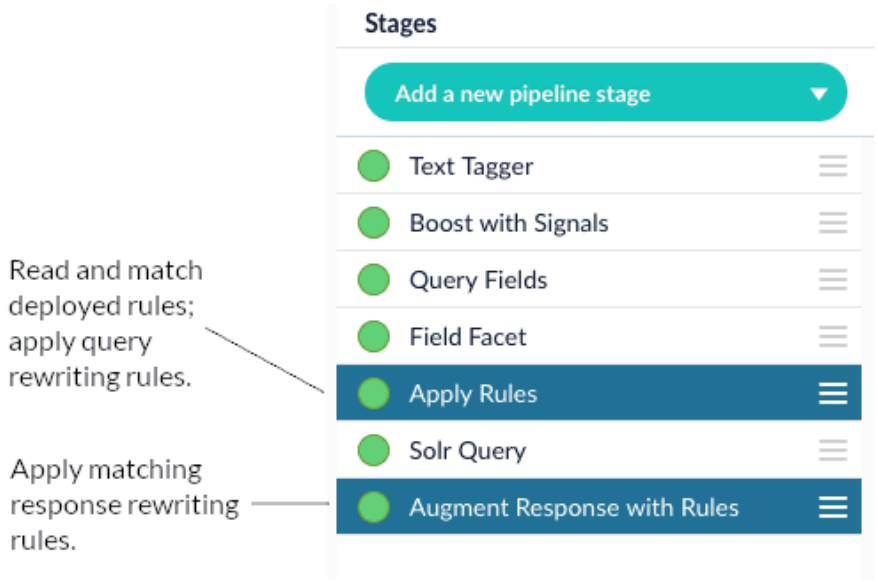
Fusion currently supports the use of the Query Elevation Component (QEC) with boost lists and filter lists. You must configure Solr in order to enable the QEC option.

For instructions about using the QEC, see *Use the Query Elevation Component*.

Additional details

Query pipeline stages for rules

These stages are part of the *default query pipeline*:



- *Apply Rules*
  This stage looks up rules that have been deployed to the `COLLECTION_NAME_query_rewrite` *collection* and matches them against the query. Matching rules that perform *query rewriting* are applied at this stage, while matching rules that perform *response rewriting* are applied by the *Modify Response with Rules* stage later in the pipeline.

	To trigger a rule that contains a tag, specify the tagname in the request URL of the user search app. See *Easily define triggers in tags* for more information.

- *Modify Response with Rules*
  Most rules operate on the request, but some *rule types*, such as banner rules or redirect rules, do their work when the response comes back. The Modify Response with Rules stage applies those rules to the response. For example, a banner rule can add a banner URL to the response before returning it to the client.

Generally, if you are using rules, you need both of these stages enabled in your query pipeline. The Apply Rules stage must come *before* the Solr Query stage, while the Modify Response with Rules stage must come *after* the Solr Query stage. Disabling or removing the Apply Rules stage will disable rules entirely. Disabling or removing the Modify Response with Rules stage will disable only the rules that perform *response rewriting*, if any.

Rules on response signals

Response signals capture a list of rule IDs that match the query in a multi-valued field named `rule_ss`.

This allows for downstream analysis on rule activity using SQL, App Insights, or a *custom Python job*.

Rules and experiments

The easiest way to integrate rules and experiments is to create one query pipeline with rules enabled and one without. Keep all other settings constant between the two pipelines. The pipeline without rules enabled must be configured manually.

To create an experimental rule, use tags on the rule and set the tag in one of the variant pipelines.

FAQ

What are arbitrary values?
Arbitrary values are values that are not actively used by Fusion. Instead, they are used by the user or the search application. For example, the **Description** is considered an arbitrary value, as it serves only to inform the user what the business rule is used for.

Additional resources

- View our training on Query Rewriting with Business Rules and Search Rewrites *(https://academy.lucidworks.com/query-rewriting-with-business-rules-and-search-rewrites)*.

General Configurations

General configurations are used to characterize a rule, including its name, description, group, and more. They are also used to set a rule's precedence, or priority over other rules.

Configuration options

Required	Parameter	Description	Example
✅	Name	A user-friendly name for the rule.	`headphones-sale-01`
	Description	A description of the rule for the person who manages business rules. This is an arbitrary value which has no effect on the rule's purpose or execution.	`"Rule for the upcoming sale on headphones. Boosts limited stock items."`
	Rule Group	See *Rule Group* for more information.	
	Tags	See *Tags* for more information.	`browse-rule`, `search-rule`
	Precedence	Determines when rules should take effect in relation to other triggered rules. A higher number means higher precedence. For example, a rule with precedence of `10` will take effect before a rule with precedence of `3`.	`10`
✅	Enabled	Determines whether the rule is enabled. If set to `false`, the rule will not fire under any conditions.	`true`

Rule group

Rule groups are used for logically grouping rules for interpretation and debugging purposes. The rule group is specified in the debug response. It does not affect which rules fire.

Tags

Tags are assigned in the UI. By using tags, you can:

- **Quickly sort rules.** Tags appear in the facet options of the Business Rules editor for fast sorting. Click a tag facet to see all rules associated with the tags.
- **Easily define triggers.** Triggers are associated with tags to determine the conditions under which the rules fire. This reduces the number of triggers needed and speeds up the development process.
  To target which rules fire, you can specify one or more tagnames in the request URL of the user search app. The rule containing the matching tagname is triggered. For example:
  - One pin rule is defined with "pin the document titled **Brussel Sprouts** when someone searches for garden", and it is tagged with `vegetable`.
  - A second pin rule is defined with "pin the document titled **Tomato** when someone searches for garden", but it is *not* tagged with `vegetable`. Instead, this rule is tagged with `fruit`.
  - A third pin rule is defined with "pin the document titled **Vegetables** when someone searches for garden", and it is tagged with `vegetable`.
    If the request URL is:
    `https://FUSION_HOST:FUSION_PORT/api/apps/APP_NAME/query/QUERY_PROFILE?q=garden&tags=vegetable`, then only the first and third rules are fired.
    To fire all three rules, the request URL is:
    `https://FUSION_HOST:FUSION_PORT/api/apps/APP_NAME/query/QUERY_PROFILE?q=garden&tags=vegetable,fruit`.

- **Separate rules by development stage.** You can add the tags `development`, `testing`, and `production` to distinguish between rules in various stages of development.
- **Define different search experiences.** In some cases, you may want to define a different experience for users who browse and users who search. Keep track of these experiences by tagging the rules accordingly.

Tags are also included in the response results:

```
{ "response" : { "numFound" : 213, "start" : 0, "maxScore" : 0.0, "docs" : [ ... ]
}, "facet_counts" : { "facet_fields" : { "tags" : { "development" : 1, "testing" :
3, "production" : 2, } }, "facet_ranges" : { } } }
```

Conditions

Conditions, or triggers, define the environment necessary for the rule action to fire. Any combination of conditions (date, field value, query, and query profile) can be defined for a rule.

Query profile triggers can be tested in the *Query Workbench* or with the *Query Pipelines API* by setting `queryProfileID` parameter on the request.

For more information, see *Business Rules Conditions Concepts*.

Configuration options

Condition types

Dates

Required	Parameter	Description	Example
✅	**Dates**	The date and time range for the rule.	`01/01/2021 12:00 to 01/31/2021 12:00`

Field value

Required	Parameter	Description	Example
✅	**Filters**	The filters that determine whether the rule fires: • Field name. The name of the field that the rule action applies to. The documents in the datasource determine what fields are available. • Field value. The values of the field that the rule action applies to. The field determines what field values are available.	Field name: `color_group_name` Field value: `red`

Query

Required	Parameter	Description	Example
✅	**Match query using**	Method used for query matching. Choose from the following: • `keywords`. The query must match the keywords of the **Search terms** value exactly. For example, if the **Search terms** value is `red pants`, only that value triggers the rule. The rule is not triggered if the value is `red`, `pants`, or `small red pants`. • `phrase`. The search query must contain one of the **Search terms** values as a subphrase, and in the same order. For example, if the **Search terms** value is `yellow shirt`, the rule is triggered by `yellow shirt` and `small yellow shirt`. The rule is not triggered if the value is `yellow small shirt`. • `text`. At least one of the **Search terms** values must match the query, but unlike `phrase`, the query only needs to match part of the value. For example, if the **Search terms** value is `green hat`, the rule is triggered by `green hat`, `green`, `hat`, and `large green hat`.	`keywords`
✅	**Search terms**	Search terms that will trigger the rule. Terms can be words or phrases.	`sale`, `shoes`, `red`

Query profile

Required	Parameter	Description	Example
✅	**Query profile ID**	The ID of the query profile that allows the rule to fire. Accepts multiple values.	`myFusionApp`

Actions

Business rule actions define what actions are taken when a rule fires.

	Some rule actions are only available to specific version of Fusion.

Configuration options

Banner

The Banner action returns an image outside of the search results that can be displayed in the frontend.

For example, an e-commerce website may display a banner to promote a sale or new product release.

Required	Parameter	Description	Example
✅	**Banner URL**	The URL of the desired banner image.	`https://example-site.com/assets/promotion.png`
	Banner zone	An arbitrary value used by the frontend, generally to instruct the search application where to place the banner image.	`TOP`

Block list

The Block List action blocks the associated values from appearing in the search results.

For example, a workplace may block documents that the user does not have permission to view.

Required	Parameter	Description	Example
✅	**Field name**	The name of the field that the rule action applies to. The documents in the datasource determine what fields are available.	`title`
✅	**Field values**	The values of the field that the rule action applies to. The field determines what field values are available.	`Voyager Jacket`

Boost attributes

The Boost Attributes action boosts documents with specific attributes in a search. Boosts can be additive or multiplicative.

For example, an e-commerce website may boost the laptop manufacturer's name if the search query contains a manufacturer.

Required	Parameter	Description	Example
✅	**Boost param**	• `bq`. An additive boost that is added to the main query's score. • `boost`. A multiplicative boost based on document scores. This boost increases or decreases the score by a relative amount.	`boost`

✅	**Boost query**	The query that the boost will apply to.	`bq`: `category_s:"Small Business"^5`
			
			`boost`: `category_s:"Small Business"`

Boost list

The Boost List action boosts documents with a large multiplicative boost so the documents appear at the top of the results. Larger boosts can override this action.

For example, an e-commerce website may boost new or seasonal items.

Required	Parameter	Description	Example
✅	**Field name**	The name of the field that the rule action applies to. The documents in the datasource determine what fields are available.	`title`
✅	**Field values**	The values of the field that the rule action applies to. The field determines what field values are available.	`Voyager Jacket`

Bury list

The Bury List action buries documents with the selected values. This action minimizes certain documents without blocking them.

For example, a workplace may bury out-of-date documents.

Required	Parameter	Description	Example
✅	**Field name**	The name of the field that the rule action applies to. The documents in the datasource determine what fields are available.	`title`
✅	**Field values**	The values of the field that the rule action applies to. The field determines what field values are available.	`Voyager Jacket`

Filter list

The Filter List action applies filter queries *(https://solr.apache.org/guide/common-query-parameters.html)* (`fq`) to the query and changes the results so a pre-selected set of content displays.

For example, an e-commerce website may display only movies rated G or PG when a user searches for children's movies.

Required	Parameter	Description	Example
✅	**Field name**	The name of the field that the rule action applies to. The documents in the datasource determine what fields are available.	`title`
✅	**Field values**	The values of the field that the rule action applies to. The field determines what field values are available.	`Voyager Jacket`

Ingroup actions

Ingroup actions perform a specific action within a *product group*. You can boost, bury, or pin specific items within a group.

Ingroup actions require a grouping field to perform the action within.

Ingroup boost list

The Ingroup Boost List action boosts an item within a product group. This action does not boost the product group.

Required	Parameter	Description	Example
✅	**Field name**	The name of the field that the rule action applies to. The documents in the datasource determine what fields are available.	`title`
✅	**Field values**	The values of the field that the rule action applies to. The field determines what field values are available.	`Voyager Jacket`
✅	**Grouping field**	The name of the group field that the rule action applies to. This group field *must* have a `group-field` parameter or the rule does not apply.	`color_group_name`

Ingroup bury list

The Ingroup Bury List action buries an item within a product group. This action does not bury the product group.

Required	Parameter	Description	Example
✅	**Field name**	The name of the field that the rule action applies to. The documents in the datasource determine what fields are available.	`title`
✅	**Field values**	The values of the field that the rule action applies to. The field determines what field values are available.	`Voyager Jacket`
✅	**Grouping field**	The name of the group field that the rule action applies to. This group field *must* have a `group-field` parameter or the rule does not apply.	`color_group_name`

Ingroup pinned

The Ingroup Pinned action sticks a document to a specific location within a product group. This ensures that users view a result in a specified place. This action does not pin the product group.

Required	Parameter	Description	Example
✅	**Field name**	The name of the field that the rule action applies to. The documents in the datasource determine what fields are available.	`title`
✅	**Field values**	The values of the field that the rule action applies to. The field determines what field values are available.	`Voyager Jacket`
✅	**Grouping field**	The name of the group field that the rule action applies to. This group field *must* have a `group-field` parameter or the rule does not apply.	`color_group_name`
✅	**Pin within a specific group value**	The values of the grouping field that the rule action applies to. The field determines what field values are available. Only a single group value is supported.	`red`
✅	**Pinned documents**	The individual documents that the rule action applies to.	`Summit Pants (red)`

Pinned

The Pinned action sticks a document to a specific location within the search results. This ensures that users view a result in a specified place.

For example, a workplace may pin documents answering frequently asked questions.

Required	Parameter	Description	Example
✅	**Field name**	The name of the field that the rule action applies to. The documents in the datasource determine what fields are available.	`id`
✅	**Pinned documents**	Individual documents to pin. Specify the following: • Value. The values of the field that the rule action applies to. The field determines what field values are available. • Position. Position within the results where the document is pinned, with `1` being the 1st result. Each position is available to only a single document.	Value: `1000026` Position: `4`

Redirect

The Redirect action displays a message at the top of the search results that the frontend can utilize to redirect a user to a specific URL. This action guides users to the desired URL when the search query isn't included in the collection.

For example, a workplace may create a redirect action that displays a link to the internal help desk when an employee searches for "computer broken".

Required	Parameter	Description	Example
✅	**Redirect**	The URL that the user will redirect to.	`https://www.activate-conf.com/`

Response value

The Response Value action sends an arbitrary value to the frontend or pipeline, which the frontend of pipeline can use to perform another action. The Response Value action can be combined with other actions within the same rule.

For example, an e-commerce website may use the response value action to display an advertisement.

Required	Parameter	Description	Example
✅	**Field name**	The name of the field that the response value is found under.	`ad-code`
✅	**Field value**	The response value that is included in the results response.	`AD_CODE_99`

Set facets

The Set Facets action customizes the facets returned in the query response. These facets are different from the facets set in the Query Workbench.

For example, an e-commerce website may display different facets on the homepage, the primary search page, and category pages.

Required	Parameter	Description	Example
✅	**Position**	Position of the facet field in the list of facets, with `1` being the top of the list. Each position is available to only a single facet.	`3`
✅	**Facet field**	The name of the facet field.	`Color`
✅	**Facet type**	The type of facet: `field` - A field found in the datasource. `range` - An accepted range for the facet field to exist within. Specify the following: • Range start. The lower bounds of the facet range. For example, `0`. • Range end. The upper bounds of the facet range. For example, `50`. • Range gap. The increment amount dividing the range, starting from the range start and ending after the range end is encompassed. For a facet range of `1–50` and a range gap of `20`, 3 facets are available: `0–20`, `20–40`, `40–60`.	`field`
Advanced options for `field` facet types			
	Select type	Determines whether the user can select a single facet or multiple facets.	`multi`
	Facet sort	Sort order of the individual facets within the facet field: • `count`. Facets are sorted by the number of documents included in the facet in descending order. • `index`. Facets are sorted in alphanumeric order.	`count`
	Min count	Sets a minimum requirement for the number of documents included in the facet in order for the facet to display.	`10`
	Boost values	Boost individual facets within the facet field in the order they are added. This allows you to override the default ordering as determined by the **Facet sort** option. For example, you may want to boost `tshirt` and `shorts` over `hoodie` during summer months.	`tshirt`
	Bury values	Bury individual facets with the facet field in the order they are added.	`hoodie`
	Suppress values	Prevent individual facets from appearing within the facet field.	`coat`
Advanced options for `range` facet types			
	Select type	Determines whether the user can select a single facet or multiple facets.	`multi`
	Facet sort	Sort order of the individual facets within the facet field: • `count`. Facets are sorted by the number of documents included in the facet in descending order. • `index`. Facets are sorted in alphanumeric order.	`count`
	Min count	Sets a minimum requirement for the number of documents included in the facet in order for the facet to display.	`10`

Set Params

The Set Params action sets a parameter name and value *(https://solr.apache.org/guide/common-query-parameters.html)* to a search query.

For example, a workplace may use the name and value to display only specific filetypes in the results.

Required	Parameter	Description	Example
✅	**Parameter name**	The name of the query parameter *(https://wiki.apache.org/solr/QueryParametersIndex)*.	`fl`
✅	**Parameter value**	The value of the query parameter.	`discountAmount`
✅	**Update policy**	The manner in which the parameter affects the results response: • `append`. Add the parameter to the default parameters in the results response. • `default`. Add the parameter to the default parameters in the results response if it is not already included in the request or the default specification in the Solr config file solrconfig.xml. • `remove`. Remove the parameter from the results response. • `replace`. Replace the default parameter in the results response.	`append`

JSON Blob

The JSON Blob action returns an *arbitrary JSON blob* to your frontend when a business rule fires. The JSON blob displays outside of the search results.

Required	Parameter	Description	Example
✅	**Blob Type**	An arbitrary value used by the frontend, generally to indicate what kind of data the JSON blob contains.	`default`
✅	**JSON Blob**	An arbitrary JSON value used by the frontend. When the rule is triggered, the JSON is sent with the results response. This behaves similarly to the **Banner** rule, given that it returns the JSON object *outside* of the search results.	`{"parameter1":"value", "parameter2": ["value_a", "value_b"]}`

Additional configuration options

- **Set response values.** Combine **Response Value** rules with other rules for purposes as *described above*.
- **Use Query Elevation Component (QEC).** Apply the rule action to configure the top results regardless of the document's score. However, enabling QEC does not boost the document's score. The QEC only supports elevation through the `id` field. If QEC is enabled but is not configured correctly in Solr, the rule does not fire.
  To use the Query Elevation Component, you must first *configure your Solr cluster*. Note that query elevation *does not boost scores*.

The chart below shows additional configuration support for each business rule action:

Action	Response value	Query Elevation Component
Banner	✘	✘
Block List	✅	✘
Boost Attributes	✅	✘
Boost List	✅	✅
Bury List	✅	✘
Filter List	✅	✅
Ingroup Boost List	✅	✘
Ingroup Bury List	✅	✘
Ingroup Pinned	✅	✘
Pinned	✅	✅
Redirect	✘	✘
Response Value	✅	✘
Set Facets	✅	✘
Set Params	✅	✘

Custom Rule Actions

Custom rule actions allow you to define a unique action to take when certain conditions are met. They involve three primary components: a query pipeline, a custom rule type, and a rule that is triggered by conditions defined by the user.

To learn use custom rule actions with the API, see *Create Custom Rule Actions with the API*.

Configuration options

Required	Parameter	Description	Example
✅	`id`	A unique ID for the custom rule type. The `id` value must be unique across all apps in Fusion (global namespace).	`custom-rule-type`
✅	`pipeline_id`	The ID of the pipeline that is invoked during rule processing.	`custom-rule-pipeline`
✅	`display_type`	A user-friendly name for the custom rule type.	`My Custom Rule`
Rule properties, included as part of the `schema`, define the values required from the user to create a rule that uses the custom rule type.			
✅	`custom_param1`	The ID of the rule's parameter. This field is used in the API but is not visible in the Rules Editor.	`1st-rule-parameter`
✅	`type`	The type of user-input.	`string`
✅	`title`	A user-friendly name for the parameter.	`Manufacturer`
✅	`description`	A user-friendly description of what the field is used for. The `description` value is shown in the field's tooltip in the Rules Editor.	`The manufacturer of the product this rule action applies to.`

Head/Tail Analysis

Head/tail analysis is:

- Also known as the Underperforming Query Rewriting feature
- Uses signals data to identify underperforming queries
- Suggests improved queries that could produce better conversion rates

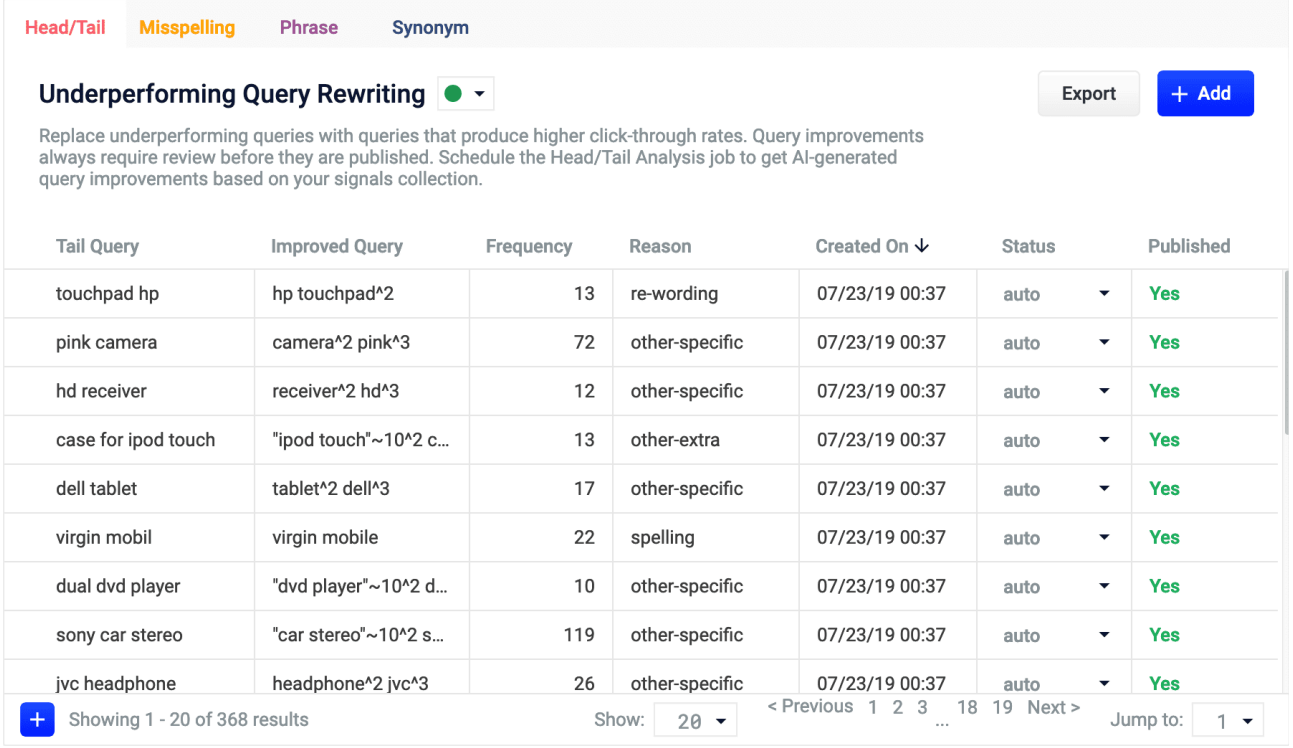
When an incoming query contains a matching underperforming query, the original query is replaced by an improved query. These improvements can be:

- Suggested by the *Head/Tail Analysis Spark job* operating on your signals data
- Created manually using the *Rules Editor* or underlying API

Query improvements are applied in the *Text Tagger stage* of the query pipeline.

The *Head/Tail Analysis job* automatically creates query improvements based on your AI-generated data. When you navigate to **Relevance** > **Rules** > **Rewrite** and select the **Head/Tail** tab, the application displays the **Underperforming Query Rewriting** screen.

Job-generated query improvements are always assigned an initial status of "Pending", never "Auto". Query improvements must be explicitly approved and published in order to be copied to the `COLLECTION_NAME_query_rewrite` collection.



Reviewing auto-generated query improvements

Query improvements that are automatically generated by the *Head/Tail Analysis job* are assigned the following status value:

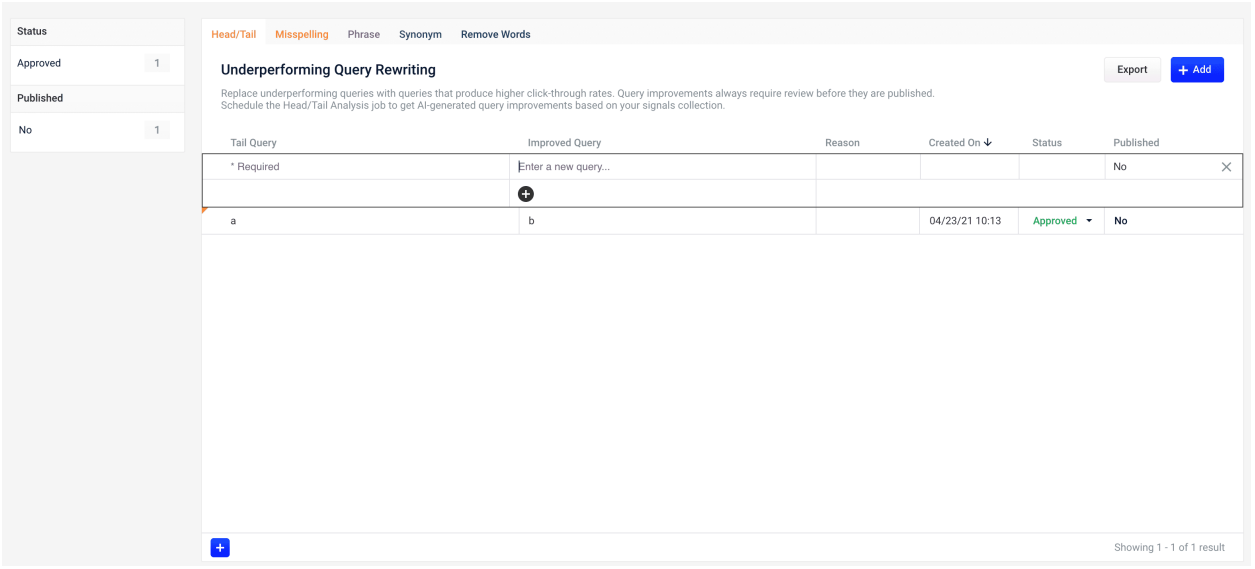
- **Auto**
  All results will be automatically deployed to the `_query_rewrite` collection.
  No action is required on these results, but you can edit them if you wish.

Adding new query improvements

You can manually add query improvements in addition to any generated by the *Head/Tail Analysis job*.

How to add a query improvement

1. Navigate to **Relevance** > **Rules** > **Rewrite**.
2. Select the **Head/Tail** tab. The application displays the **Underperforming Query Rewriting** screen.
3. At the bottom of the rules list, click the ⊕ icon.
   A new query improvement appears at the top of the list:



4. Enter the underperforming query.
5. Enter one or more query improvements.
   It is not necessary to set a confidence value.
6. Select the query improvement's status, depending on whether you want to deploy it the next time you publish your changes ("Approved") or save it for further review ("Pending").
7. Click the check mark to save the new query improvement.

Publishing your changes

How to publish updated query improvements

1. In the **Underperforming Query Rewriting** screen, click the **PUBLISH** button.
   Fusion prompts you to confirm that you want to publish your changes.
2. Click **PUBLISH**.

You can un-publish a query rewrite by changing its status to "denied", then clicking **PUBLISH**.

For more information about how to use underperforming query rewrites in the *Rules Editor*, see *Use Query Rewrites in the Rules Editor*.

Query rewrite jobs post-processing cleanup

To perform more extensive cleanup of query rewrites, complete the procedures in *Query Rewrite Jobs Post-processing Cleanup*.

Template Expressions

Template expressions are a way to map field values in a document. They are used to configure some Fusion pipeline stage and messaging services. When the value of a configuration parameter is a template expression, that expression is dynamically evaluated at runtime by the StringTemplate *(https://github.com/antlr/stringtemplate4/blob/master/doc/cheatsheet.md)* library.

Fusion template expressions are delimited by angle bracket characters `<` and `>`. The expression consists of the name of a variable in the scope of that component.

Since these variables are Java objects, if object **foo** of type Foo has a field named **bar**, the expression **<foo.bar>** will evaluate to the string representation of the contents of field **bar**.

For example, given a PipelineDocument where a field named **title** has a value of **Star Wars**, the template expression `<doc.title>` evaluates to **Star Wars**.

See the reference guide for *index pipeline stage*, *query pipeline stage*, and *messaging service* template expression variables.

Index Pipeline Stage Template Expression Variables

Index stages have the following variables available:

- `ctx`. *Context*: used to pass variables between stages
- `doc`. *PipelineDocument*: Fusion internal representation of a Solr Document. Please note that if you change this class in ways that affect the serialized form (adding fields, for example) you need to change the deserialization code in `GenericJSONDocumentMessageReader`.

The format required is `<request.q>`.

Messaging Services Template Expression Variables

Messages have a set of variables which correspond to the parts of a system message:

- id
- to
- from
- subject
- body
- type
- schedule

A system message is the result of evaluating the following expression:

```
<id><to><from><subject><body><type><schedule>
```

Query Pipeline Stage Template Expression Variables

Query stages have the following variables available:

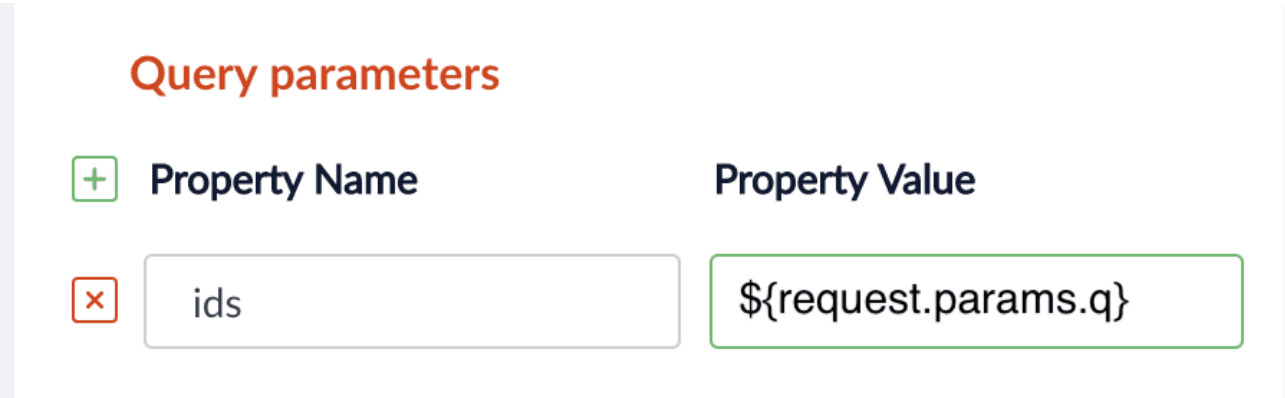
- `ctx`. *Context*: used to pass variables between stages
- `reqResp`. *QueryRequestAndResponse*: deserializes pre-request parameters and makes them available

The format required is `<request.q>`.

Example

Check that a query has an `fq` filter on a given field for a given value:

In the UI, set property name to `ids` and Property Value to `req.hasParam("fq")`



Misspelling Detection

The Misspelling Detection feature maps misspellings to their corrected spellings. When Fusion receives a query containing a known misspelling, it rewrites the query using the corrected spelling in order to return relevant results instead of an empty or irrelevant results set.

Spelling corrections are applied in the *Text Tagger stage* of the query pipeline.

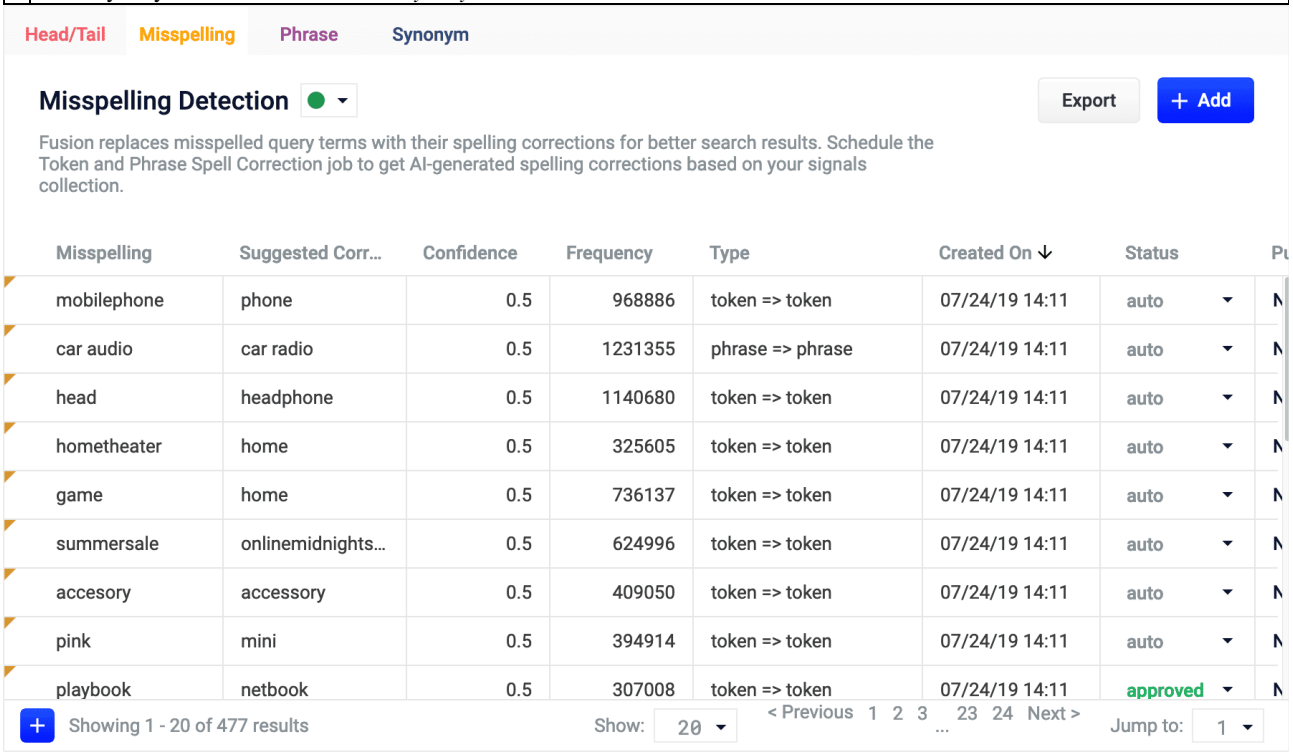
The *Token and Phrase Spell Correction job* automatically creates spelling corrections based on your AI-generated data. When you navigate to **Relevance** > **Rules** > **Rewrite** and select the **Misspelling** tab, the application displays the **Misspelling Detection** screen.

You can:

- *Review or edit* the output from the job
- *Manually add new spelling corrections*
- *Publish changes* from the `_query_rewrite_staging` collection

When you manually add new spelling corrections, subsequent job runs use those documents as input for machine learning to improve the job's output. Unlike job-generated documents, manually-added query rewriting documents are never overwritten by new job output.

Misspelled terms are completely replaced by their corrected terms. If you want to expand the query to include all alternative terms, set the synonyms to bi-directional. See *Synonym Detection* for more information.

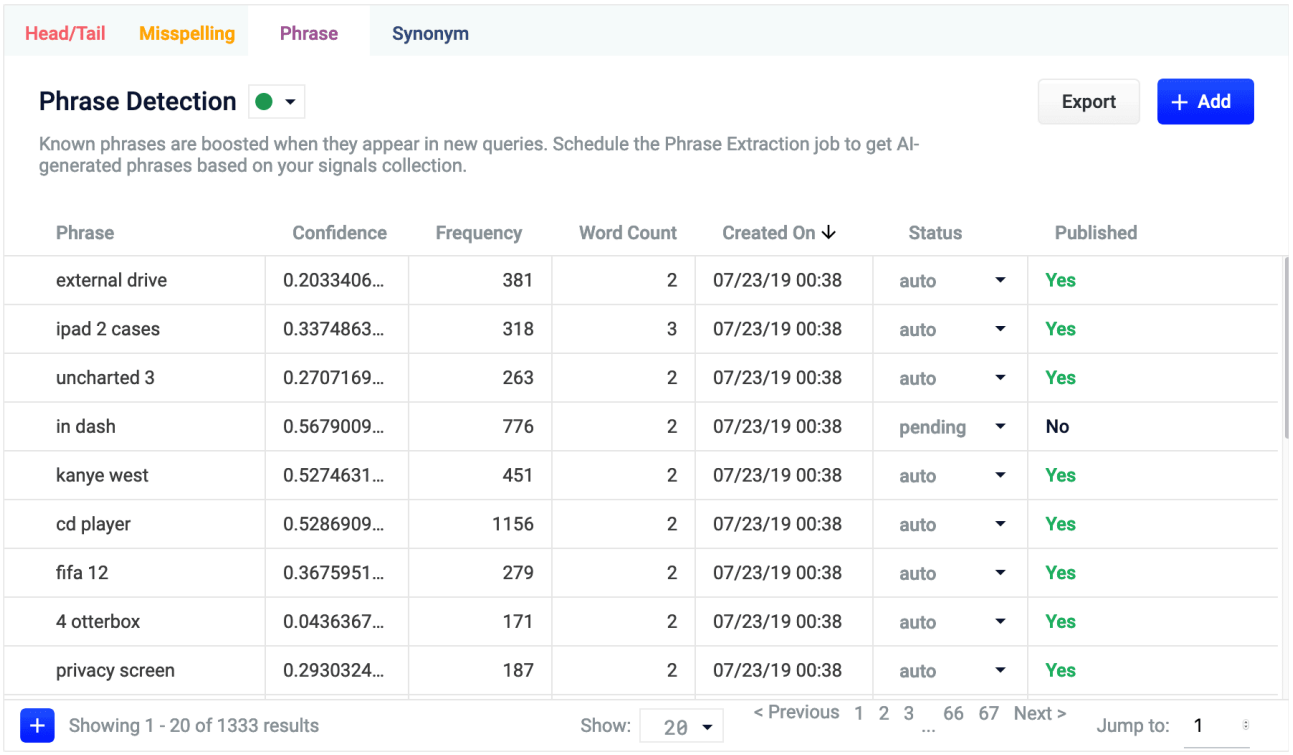


To learn how to use misspelling detection in the *Rules Editor*, see *Use Query Rewrites in the Rules Editor*.

Phrase Detection

Phrase detection identifies phrases in your signals so that results with matching phrases can be boosted. This helps compensate for queries where phrases are not distinguished with quotation marks. For example, the query `ipad case` is rewritten as `"ipad case"~10^2`, meaning if `ipad` and `case` appear **within 10** terms (whitespace-delimited tokens) of each other, then boost the result by a factor of **2**.

Phrases are applied in the *Text Tagger stage* of the query pipeline.



For more information about how to use phrase detection in the *Rules Editor*, see:

- *Use Query Rewrites in the Rules Editor*
- *Use Phrase Detection*

Query Rewriting

Query rewriting is a strategy for improving relevancy using AI-generated data. Many of Fusion's features can be used to rewrite incoming queries prior to submitting them to Fusion's Solr core. These rewrites produce more relevant search results with higher conversion rates.

For example, when spelling corrections are used for query rewriting, a misspelled query can return the same search results as a correctly-spelled query, instead of returning irrelevant results or no results. Spelling corrections are one of several available *query rewriting strategies*. Apply all available strategies for best results.

See also the *Query Rewriting API*.

Fusion can also rewrite Solr's responses before returning them to the search application; see *Response Rewriting*.

Query rewriting strategies

Fusion provides a variety of query rewriting strategies to improve relevancy:

- *Business rules*
- *Underperforming query rewriting*
- *Misspelling detection*
- *Phrase detection*
- *Synonym detection*
- *Remove Words*

With the exception of business rules, which are always manually created, these strategies correspond to certain *Spark jobs*. Lucidworks recommends configuring and scheduling all of these jobs for best results. You can also train the jobs by manually adding documents to their output. Manually-added documents are used for machine learning and are never overwritten by new job output.

Query rewriting strategies are applied in the following order:

1. **Business rules** - If a query triggers a business rule, then the business rule overrides any query rewriting strategies that conflict with it.
2. **Query rewrites**
    a. **Underperforming query rewriting** - If a query triggers an underperforming query rewrite, then this strategy overrides all subsequent query rewriting strategies.
    b. **Remove words** - To help increase the number of results returned, Remove Words query rewrites removes words from the users' search.
3. **Synonym detection**
4. **Misspelling detection *and* phrase detection** - The query rewriting results from both of these strategies are applied together. To use only the strategy with the longest surface form, you can configure the Text Tagger query stage with **Overlapping Tag Policy** set to "LONGEST_DOMINANT_RIGHT".

Business rules

Business rules are manually-created formulas for rewriting queries. This is the most versatile strategy for creating custom query rewrites. It supports a variety of conditions and actions to address a wide range of use cases. When you need a very specific query rewrite, this is the best strategy.

Business rules are applied in the *Apply Rules stage* of the query pipeline.

See *Business Rules* to learn how to create, edit, and publish business rules.

Underperforming query rewriting

Head/tail analysis is:

- Also known as the Underperforming Query Rewriting feature
- Uses signals data to identify underperforming queries
- Suggests improved queries that could produce better conversion rates

When an incoming query contains a matching underperforming query, the original query is replaced by an improved query. These improvements can be:

- Suggested by the *Head/Tail Analysis Spark job* operating on your signals data
- Created manually using the *Rules Editor* or underlying API

Query improvements are applied in the *Text Tagger stage* of the query pipeline.

See *Head/Tail Analysis (Underperforming Query Rewriting)* to learn how to review, edit, create, and publish query improvements.

Misspelling detection

The Misspelling Detection feature maps misspellings to their corrected spellings. When Fusion receives a query containing a known misspelling, it rewrites the query using the corrected spelling in order to return relevant results instead of an empty or irrelevant results set.

Spelling corrections are applied in the *Text Tagger stage* of the query pipeline.

	Misspelled terms are completely replaced by their corrected terms. If you want to expand the query to include all alternative terms, set the synonyms to bi-directional. See *Synonym Detection* for more information.

See:

- *Misspelling Detection* for general information
- *Use Misspelling Detection* for information about how to review, edit, create, and publish spelling corrections

Phrase detection

Phrase detection identifies phrases in your signals so that results with matching phrases can be boosted. This helps compensate for queries where phrases are not distinguished with quotation marks. For example, the query `ipad case` is rewritten as `"ipad case"~10^2`, meaning if `ipad` and `case` appear **within 10** terms (whitespace-delimited tokens) of each other, then boost the result by a factor of **2**.

Phrases are applied in the *Text Tagger stage* of the query pipeline.

See:

- *Phrase Detection* for general information
- *Use Phrase Detection* for information about how to review, edit, create, and publish spelling corrections

Synonym detection

The Synonym Detection feature generates pairs of synonyms and pairs of similar queries. Two words are considered potential synonyms when they are used in a similar context in similar queries. A query that contains a matching term is expanded to include all of its synonyms, with the original term boosted by a factor of two. Synonyms are applied in the *Text Tagger stage* of the query pipeline.

See:

- *Synonym Detection* for general information
- *Use Synonym Detection* for information about how to review, edit, create, and publish synonym corrections

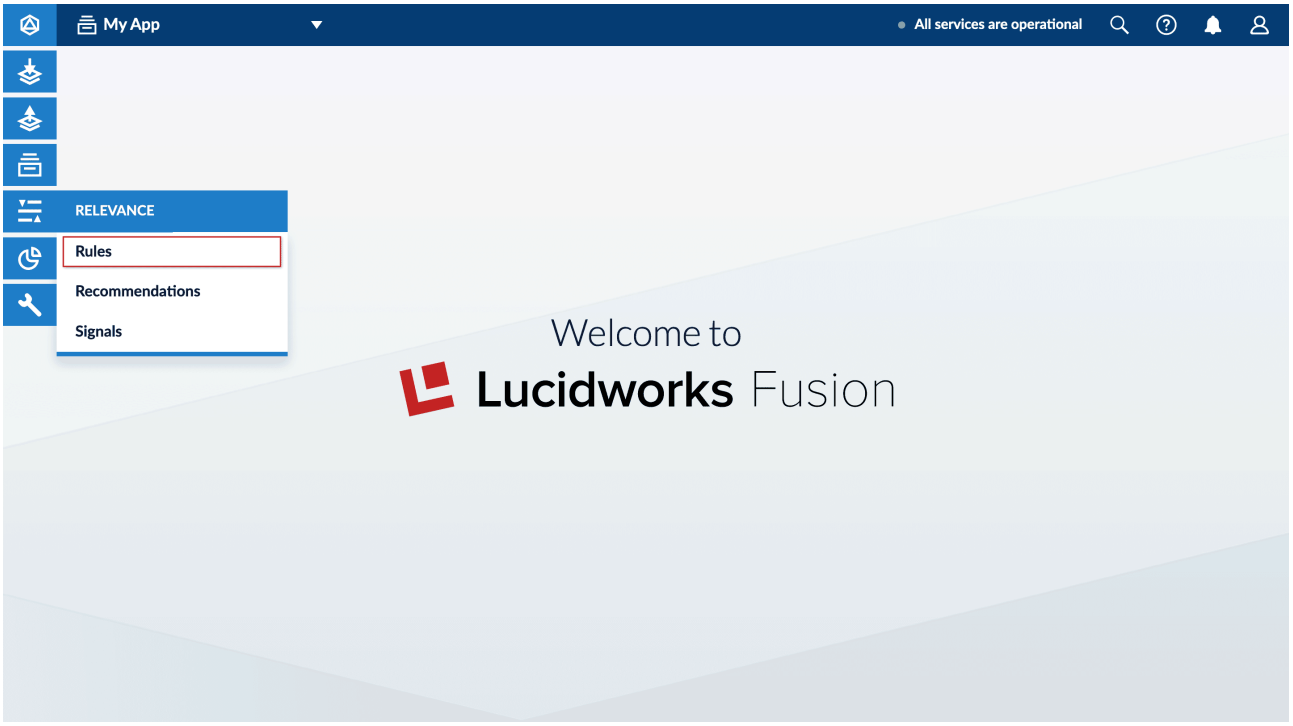
Remove words

Use a Remove Words query rewrite to remove particular phrases from queries. Unlike other rewrites, Remove Words rules are entered manually and aren't generated by a job.

See *Remove Words* to learn how to remove words from your users' searches.

Rules Editor

The Rules Editor allows you to view, edit, create, approve, enable, and publish rules powered by Fusion. Access the Rules Editor from the Fusion UI by navigating to **Relevance > Rules**:



Read more about the *Rules Editor*.

Query rewrite collections

For detailed information about query rewriting, see:

- *Manage Collections in the Fusion UI*
- *Collections*

For each app, two *auxiliary collections* are dedicated to documents used for query rewriting:

- `COLLECTION_NAME_query_rewrite_staging`
  Certain Spark jobs send their output to this collection. *Rules* are also written to this collection initially.
  Some of the content in this collection requires manual review before it can be migrated to the `COLLECTION_NAME_query_rewrite`, where query pipelines can read it. See below for details.
- `COLLECTION_NAME_query_rewrite`
  This collection is optimized for high-volume traffic. Query pipelines can read from this collection to find rules, synonyms, spelling corrections, and more with which to rewrite queries and responses.

Each app contains exactly one of each of these collections, associated with the app's default collection. They are not created again for additional collections created within the same app.

Documents move from `COLLECTION_NAME_query_rewrite_staging` to the `COLLECTION_NAME_query_rewrite` collection only when they are approved (either automatically on the basis of their confidence scores or manually by a human reviewer) *and* a Fusion user clicks **Publish**. The `review` field value indicates whether a document will be published when the user clicks **Publish**:

`review=auto`	A job-generated document has a sufficiently high confidence score and is automatically approved for publication.
`review=pending`	A job-generated document has an ambiguous confidence score and must be reviewed by a Fusion user.
`review=approved`	A Fusion user has reviewed the document and approved it for publication.
`review=denied`	A job-generated document has a low confidence score, or a Fusion user has reviewed and denied it for publication.

In the query rewriting UI, the value of the `review` field appears in the **Status** column.

You can review and approve or deny documents using the query rewriting UI. You can also change a document's status to "pending" to save it for later review.

Rules Simulator query profile

> Rules Simulator is only available for Fusion 5.x.x organizations that do *not* have a valid Predictive Merchandiser or Experience Optimizer license.

The *Rules Simulator* allows product owners to experiment with rules and other query rewrites in the `COLLECTION_NAME_query_rewrite_staging` collection before deploying them to the `COLLECTION_NAME_query_rewrite` collection.

Each app has a `COLLECTION_NAME_rules_simulator` query profile, configured to use the `COLLECTION_NAME_query_rewrite_staging` collection for query rewrites instead of the `COLLECTION_NAME_query_rewrite` collection. This profile is created automatically whenever a new app is created.

See *Configure the Rules Simulator Query Profile* for more information about configuration.

Query pipeline stages for query rewriting

These query rewriting stages are part of any default query pipeline:

- *Apply Rules query stage*
  This stage looks up rules that have been deployed to the *`COLLECTION_NAME_query_rewrite` collection* and matches them against the query. Matching rules that perform *query rewriting* are applied at this stage, while matching rules that perform *response rewriting* are applied by the *Modify Response with Rules* stage later in the pipeline.

> To trigger a rule that contains a tag, specify the tagname in the request URL of the user search app. See *Easily define triggers in tags* for more information.

- *Text Tagger query pipeline stage*
  This stage uses the *SolrTextTagger* handler to identify known entities in the query by searching the *`COLLECTION_NAME_query_rewrite` collection*.

  > For Fusion 5.x.x organizations that do *not* have a Predictive Merchandiser license, the Solr Text Tagger handler also searches the `COLLECTION_NAME_query_rewrite_staging` collection in the case of the Fusion query rewriting *Simulator*).

  The purpose of the search is to perform query rewriting using matches from the following items:
  - *Spelling corrections*
  - *Phrase boosts*
  - *Head/Tail Analysis (Underperforming query improvements)*
  - *Synonym Detection*

Spark jobs for query rewriting

This section describes how Spark jobs support query rewriting. These jobs read from the signals collection and write their output to the `COLLECTION_NAME_query_rewrite_staging` collection. High-confidence results are automatically migrated from there to the `COLLECTION_NAME_query_rewrite` collection, while ambiguous results remain in the staging collection until they are reviewed and approved. You can review job results in the Query Rewriting UI.

- *Daily query rewriting jobs* are created and scheduled automatically when you create a new app.
- *Additional query rewriting jobs* can be created manually.

> For best relevancy, enable all of these jobs.

Daily query rewriting jobs

When a new app is created, the jobs below are also created and scheduled to run daily, beginning 15 minutes after app creation, in the following order:

1. *Token and Phrase Spell Correction job*
   Detect misspellings in queries or documents using the numbers of occurrences of words and phrases.
2. *Phrase Extraction job*
   Identify multi-word phrases in signals.
3. *Synonym Detection*
   Use this job to generate pairs of synonyms and pairs of similar queries. Two words are considered potential synonyms when they are used in a similar context in similar queries.

Process flow

The first and second jobs can provide input to improve the Synonym job's output:

- Token and Phrase Spell Correction job results can be used to avoid finding mainly misspellings, or mixing synonyms with misspellings.
- Phrase Extraction job results can be used to find pairs of synonyms with multiple tokens, such as "lithium ion"/"ion battery".

The Phrase Extraction and Synonym Detection jobs are triggered by the success of the previous job: the phrase detection job runs only if the spell correction job succeeds, and the synonym job runs only if the phrase detection job succeeds.

Additional query rewriting jobs

These jobs also produce results that are used for query rewriting, but must be created manually:

- *Head/Tail Analysis job*
  Perform head/tail analysis of queries from collections of raw or aggregated signals, to identify underperforming queries and the reasons. This information is valuable for improving overall conversions, Solr configurations, auto-suggest, product catalogs, and SEO/SEM strategies, in order to improve conversion rates.
- *Ground Truth job*
  Estimate ground truth queries using click signals and query signals, with document relevance per query determined using a click/skip formula.

"rules" role for query rewriting users

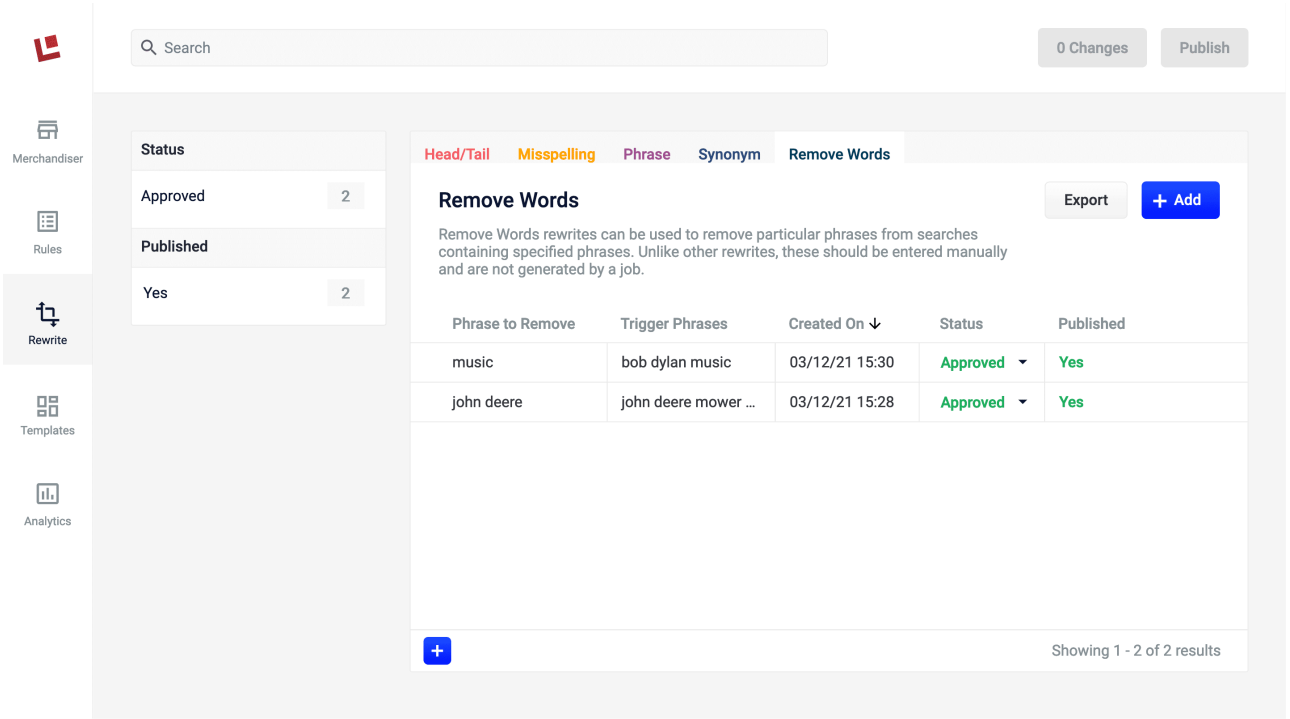
The "rules" *role* provides permissions to access query rewriting features for all Fusion apps. A Fusion admin can create a user account with this role to give a business user access to the Query Rewriting UI.

Query rewrite jobs post-processing cleanup

To perform more extensive cleanup of query rewrites, complete the procedures in *Query Rewrite Jobs Post-processing Cleanup*.

Remove Words

Use a Remove Words query rewrite to remove particular phrases from queries. Unlike other rewrites, Remove Words rules are entered manually and aren't generated by a job.



To learn how to use Remove Words and other query rewrites in the *Rules Editor*, see *Use Query Rewrites in the Rules Editor*.

Response Rewriting

Response rewriting is a strategy for improving relevancy using AI-generated data by modifying Solr's response before passing it to the search application. There are two approaches to configuring response rewriting:

Rules

Any *rule type* can add content to the response by modifying the `responseValues` field. These two rule types can replace the whole response:

- Banner
  Display a user-defined banner message when the rule fires.
- Redirect
  Send users to a different URL instead of the search results.

See the main *Rules* topic for details.

Query pipeline stages

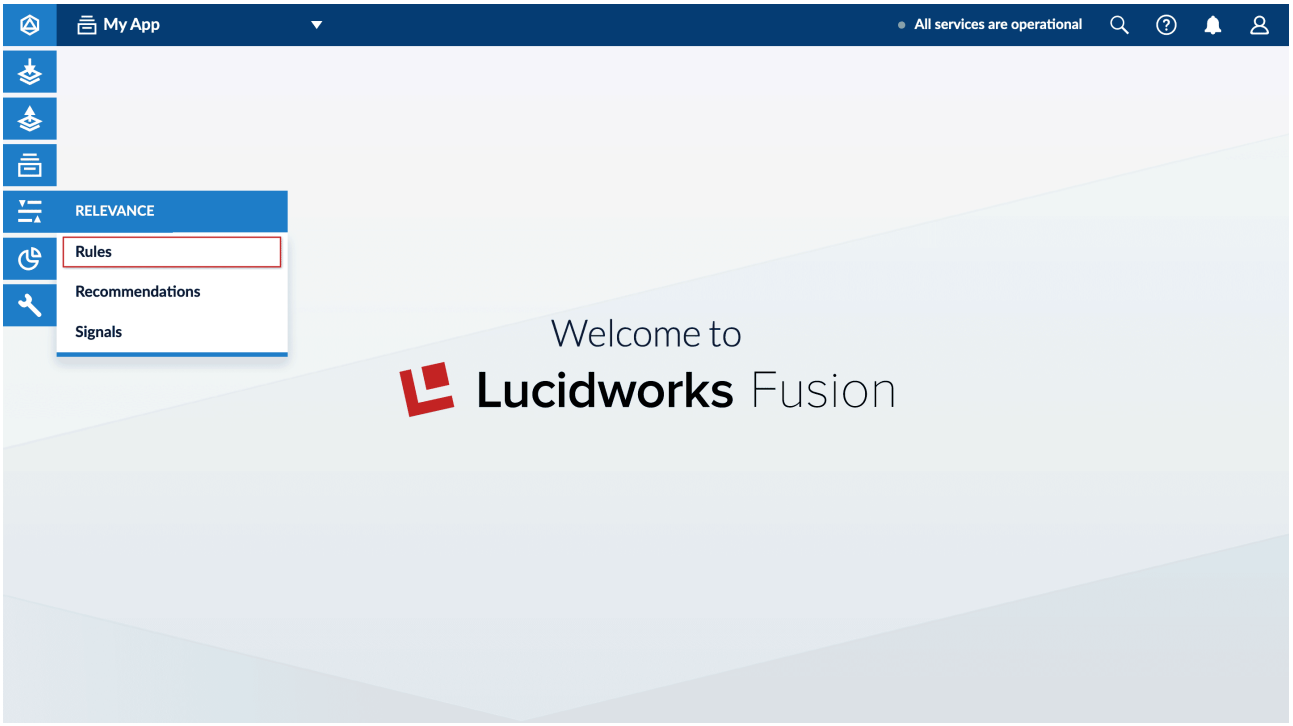
Query pipeline stages that perform response rewriting must appear *after* the *Solr Query stage*. These stages fall into two categories:

- **Stages that act on the whole set of results**
  - *Response Shuffle stage*
    "De-bias" results by shuffling the top N results randomly.
  - *Response Pairwise Swap stage*
    "De-bias" results by swapping the search results at any two positions, such as positions 1 and 2, positions 3 and 4, and so on.
- **Stages that act on individual documents**
  - *Response Document Exclusion stage*
    Drop all documents that match all of the specified rules.
  - *Modify Response with Rules stage*
    Apply *rules* to the response.

Rules Editor

Overview

The Rules Editor allows you to view, edit, create, approve, enable, and publish rules powered by Fusion. Access the Rules Editor from the Fusion UI by navigating to **Relevance** > **Rules**:



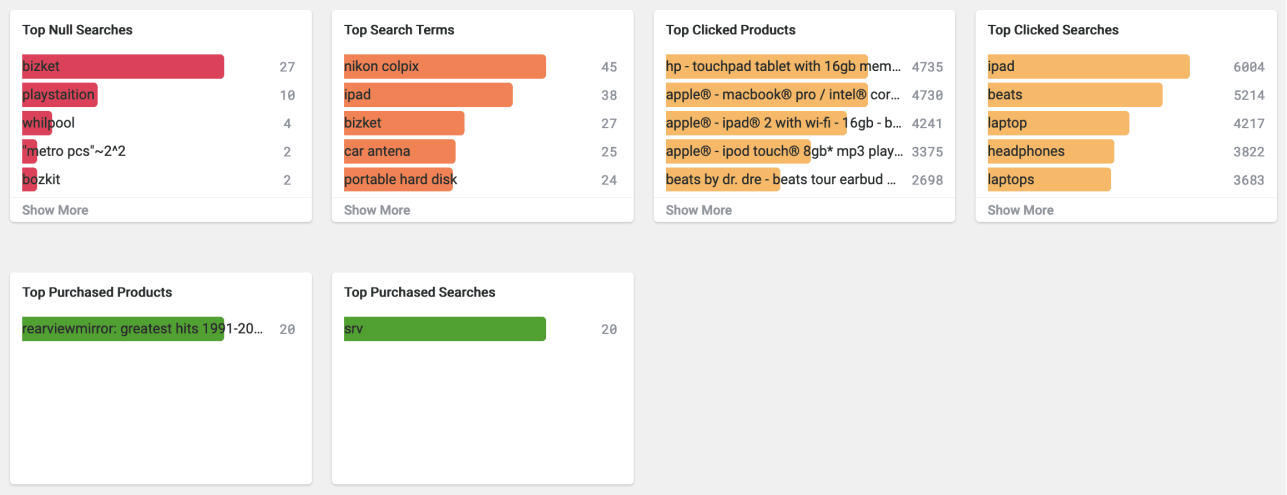
A rule's various states determine whether the rule is in effect for everyone using your search application:

State	Options	Description
Status	*Enabled* or *Disabled*	If a rule's status is *Enabled*, the rule is in effect. If a rule's status is *Disabled*, the rule is not in effect. This is only true if the rule is published, as described below.
Review	*Approved* or *Denied*	If a rule's review state is *Approved*, and you publish the changes, the latest changes to that rule are in effect. If the state is *Denied*, and you publish the changes, the Rules Manager deletes the rule. Unlike deleting the rule with the UI, if you delete the rule by publishing it while in a *Denied* review state, you cannot undo the action.
Published	*Yes* or *No*	If a rule's published state is *Yes*, the latest changes to the rule are in effect. If the state is *No*, the latest changes are not in effect. This also applies to changes to the rule's status, such as disabling to rule.

For more information:

- *How to use the Rules Editor Dashboard*
- *How to use the Rules Editor Business Rules*
- *How to use the Rules Editor Query Rewrites*

Signals data

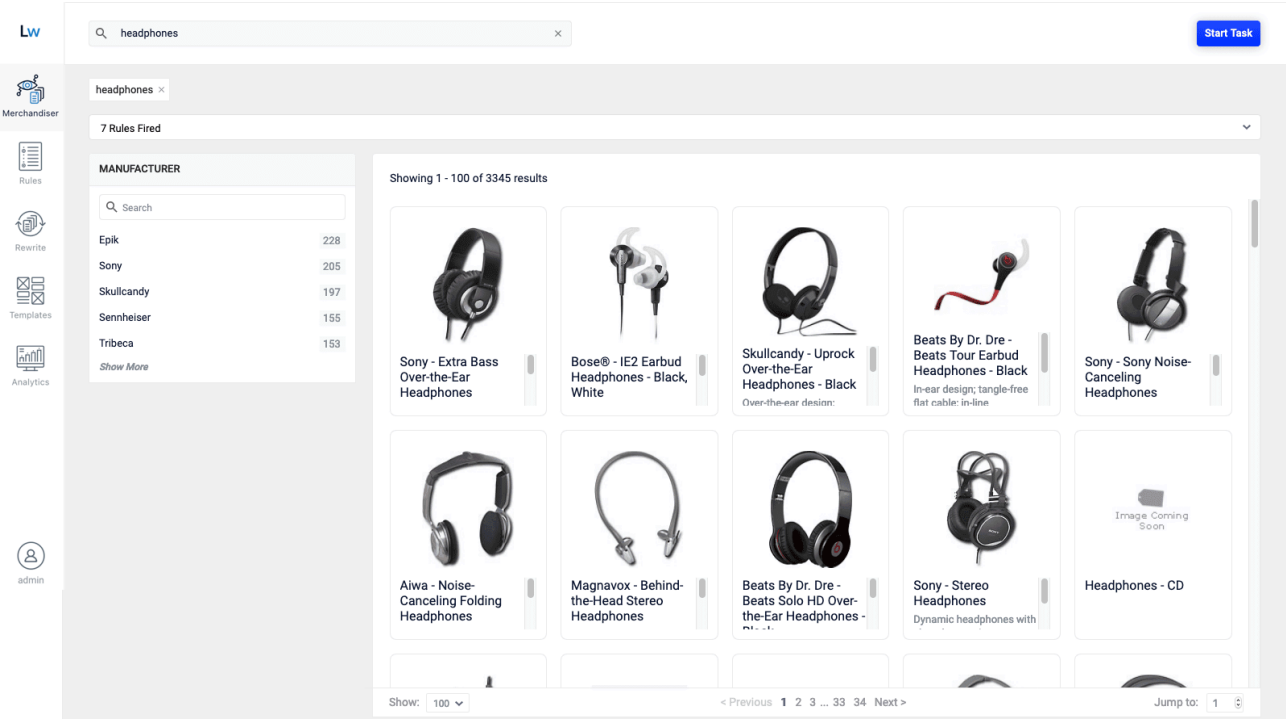


- **Top Null Searches.** The top search terms that yielded 0 results. A null search is considered a poor user experience. Use *query rewrites* to reduce the likelihood of a visitor receiving 0 results.
- **Top Search Terms.** The most commonly searched terms.
- **Top Clicked Products.** The most commonly clicked products or documents. This may include searches, recommendations, static links, or outside referrals.
- **Top Clicked Searches.** The most common searches that resulted in a click.
- **Top Purchased Products.** The most commonly purchased products. This may include searches, recommendations, static links, or referrals.
- **Top Purchased Searches.** The most commonly purchased products that were identified from a search.

	Use *query rewrites* to adapt your search results and guide visitors to top clicked and top purchased products.

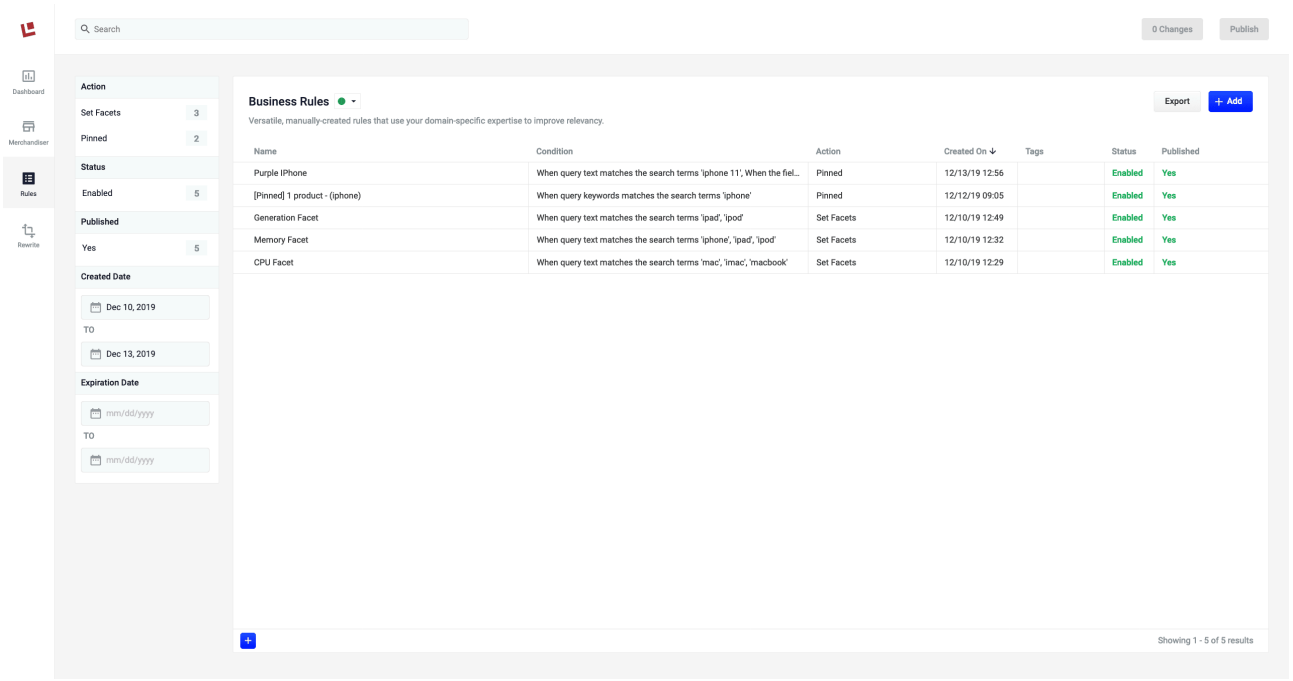
Predictive Merchandiser and Experience Optimizer

Predictive Merchandiser and Experience Optimizer are both AI-powered tools that provide insights, customer recommendations, and search optimization in a single comprehensive product.



Predictive Merchandiser and Experience Optimizer are separate offerings from Fusion. See the *Predictive Merchandiser documentation* for more information.

Rules



The Rules screen is used to create *Business Rules* within the Rules Editor.

Business rules are manually-created formulas for rewriting queries. This is the most versatile strategy for creating custom query rewrites. It supports a variety of conditions and actions to address a wide range of use cases. When you need a very specific query rewrite, this is the best strategy.

Business rules are applied in the *Apply Rules stage* of the query pipeline.

Learn how to use the *Rules Editor Business Rules*.

UI elements

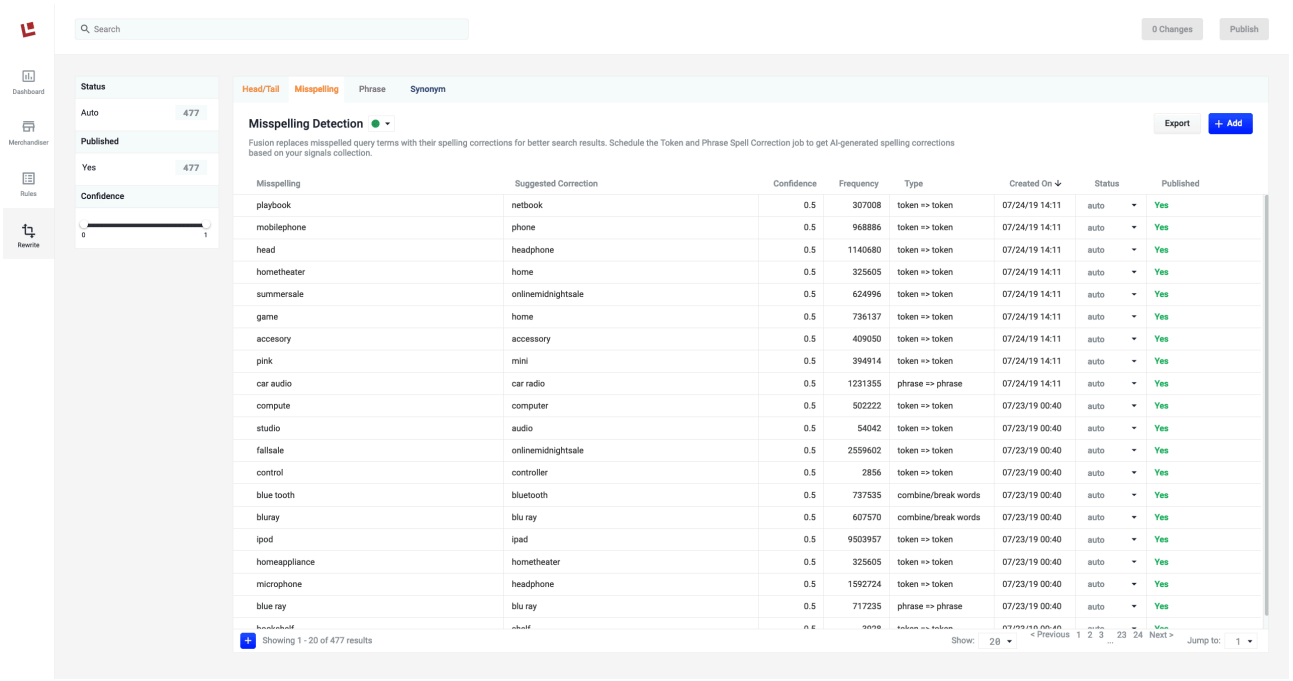
Facets

- **Action.** The *built-in rule action* for the rule.
- **Status.** Whether a rule is enabled or disabled.
- **Published.** Whether or not a rule has been published.
- **Created Date.** The date the rule was created.
- **Expiration Date.** The date the rule will expire.

Rules list

The rules list can be faceted, searched, or sorted according to the various values the rules hold.

Rewrite



Learn how to use the *Rules Editor Query Rewrites*.

UI elements

Facets

- **Status.** Whether a rule is enabled or disabled.
- **Published.** Whether or not a rule has been published.
- **Confidence.** The confidence score for the rule. A confidence score is automatically set by Fusion. It can be manually set for user-created rules. The confidence score is used, in part, to decided whether a rule is automatically published or not. See *Query Rewriting concepts* for more information.

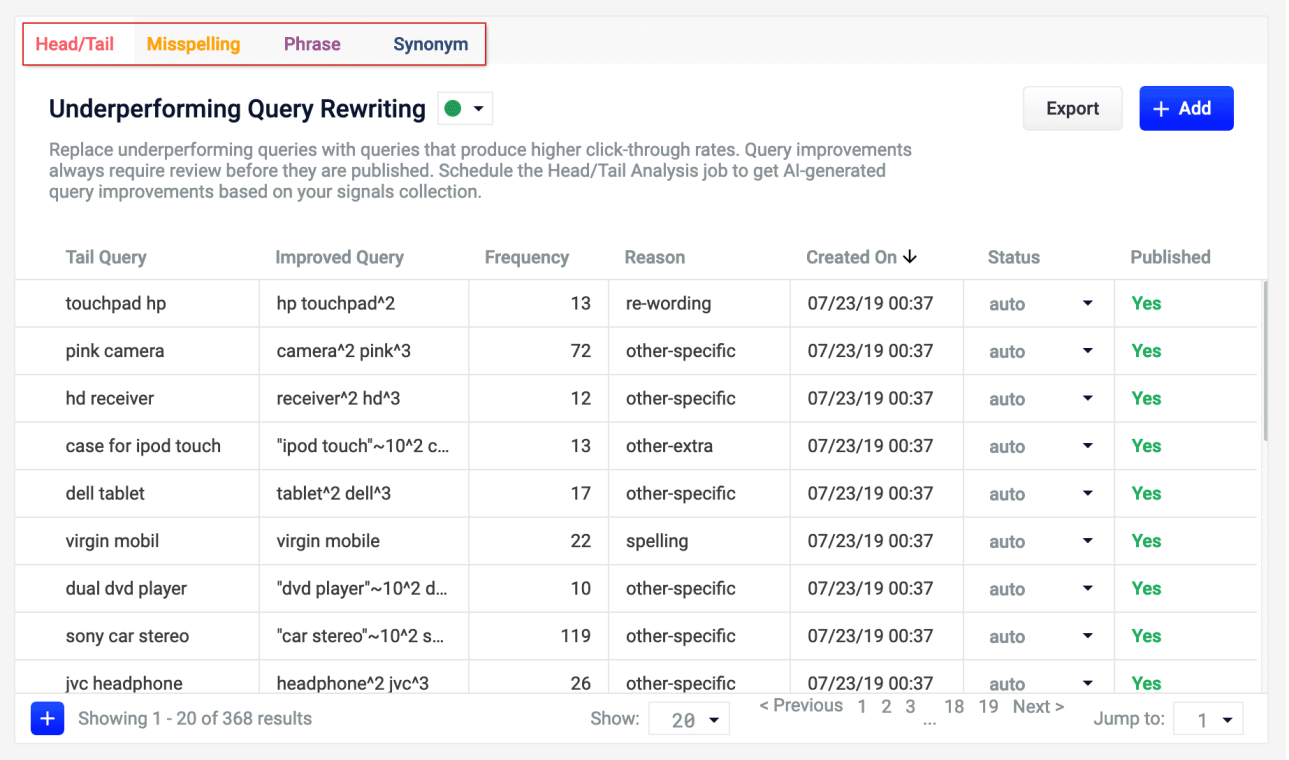
	Confidence is not used for Head/Tail query rewrites.

Rules list

You can facet, search, or sort the rules list according to the values the rules hold.

Rule type selectors

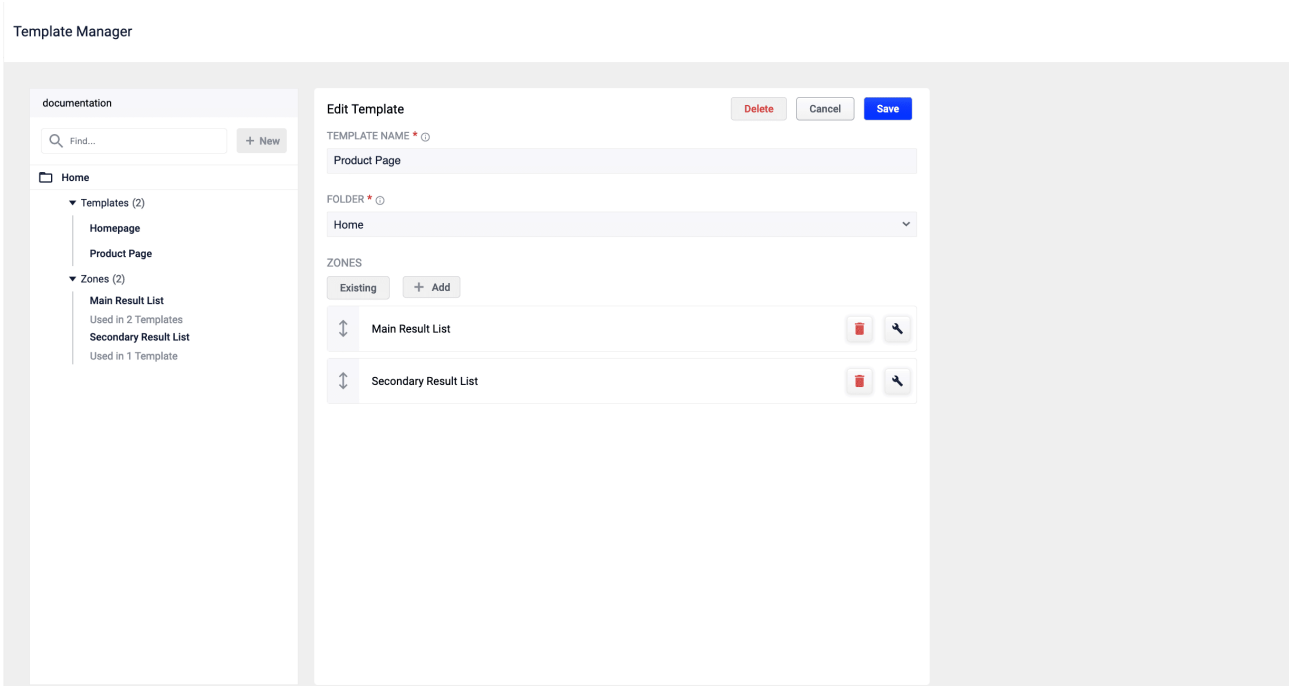
There are buttons at the top of the rules list that allow you to choose what query rewrite rule types you would like to work with:



- **Head/Tail.** Replace underperforming queries with queries that produce higher click-through rates. Query improvements always require review before they are published. Schedule the Head/Tail Analysis job to get AI-generated query improvements based on your signals collection. See *Underperforming Query Rewriting*.
- **Misspelling.** Fusion replaces misspelled query terms with their spelling corrections for better search results. Schedule the Token and Phrase Spell Correction job to get AI-generated spelling corrections based on your signals collection. See *Misspelling Detection*.
- **Phrase.** Known phrases are boosted when they appear in new queries. Schedule the Phrase Extraction job to get AI-generated phrases based on your signals collection. See *Phrase Detection*.
- **Synonym.** Synonyms expand queries so that they include all known synonyms of the query terms. Schedule the Synonym Detection job to get AI-generated synonyms based on your signals collection. See *Synonym Detection*.
- **Remove Words.** Remove particular phrases from queries. Unlike other rewrites, these should be entered manually and are not generated by a job. See *Remove Words*.

Templates

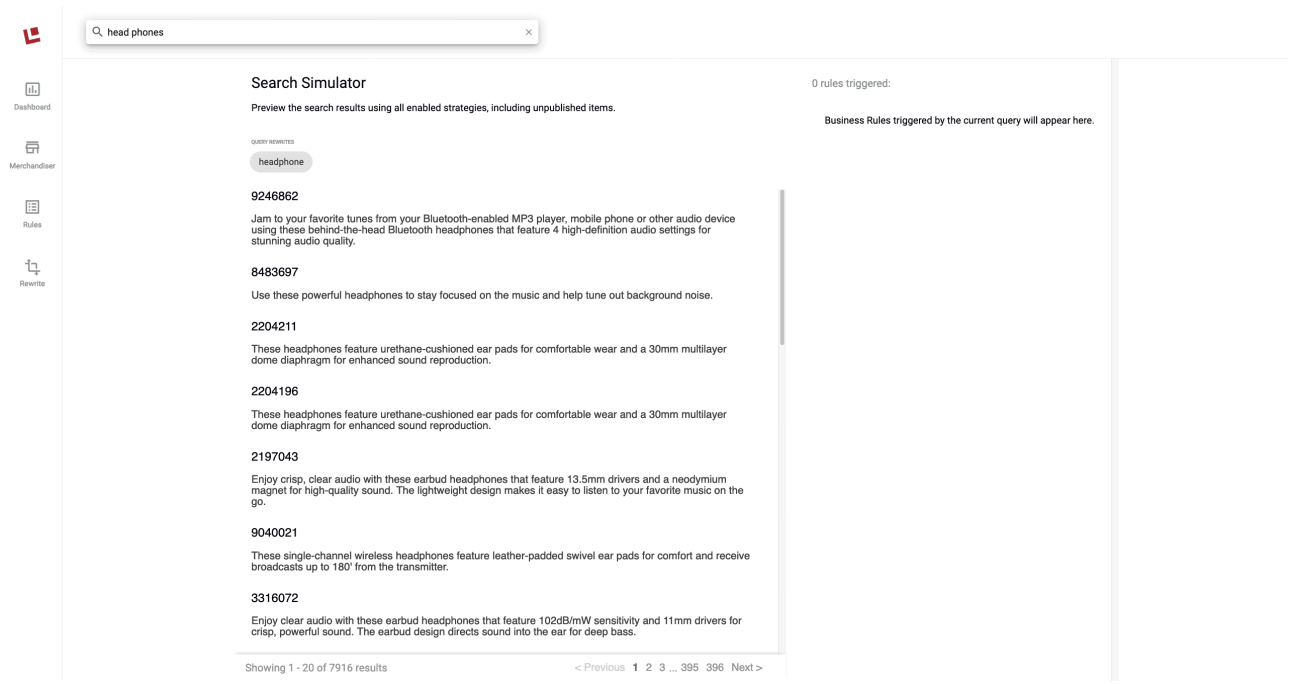
The Templates screen in the Rules Editor allows you to design, test, and implement a wide variety of search experiences. Results are conditionally delivered from multiple different query pipelines, enabling you to use Fusion's search, browse, and AI-driven functionality throughout your site.



Templates are only available with a valid license for *Predictive Merchandiser*. If available, you will see the templates icon in the left navigation panel:

Simulator

The Simulator provides an interactive preview of how your staged rules affect relevancy, using your search data and a simple search interface. When you enter a query, the Simulator shows you the triggered rules, in the order in which they were triggered, along with any triggered facets.

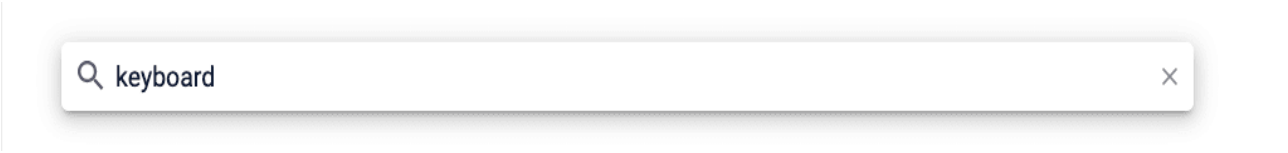


You can edit any of the triggered rules, then re-run the query to see new results.

The Simulator sends requests to the `COLLECTION_NAME_rules_simulator` query profile, which you can configure to point to any pipeline and collection in your app.

The Simulator is only visible in the Rules Editor if you *do not* have a valid license for *Predictive Merchandiser*. If available, you will see the simulator icon in the left navigation panel.

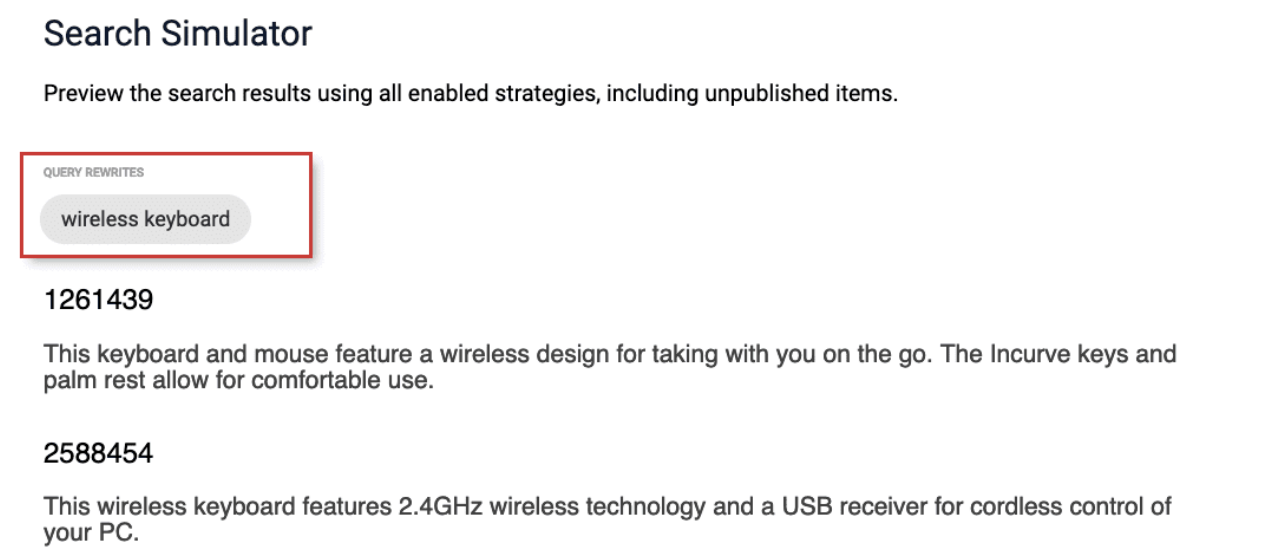
Search



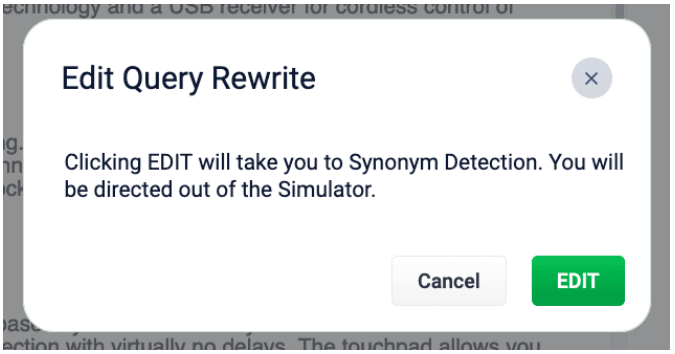
Use the search bar in the top nav panel to simulate, or preview, your staged rules.

Rule indicators

After entering a search query, the active rules affecting the results are displayed above the results. For example, a query for the term `keyboard` with a query rewrite rule would display an indicator for the rewritten query `wireless keyboard` in the left panel of the Simulator:

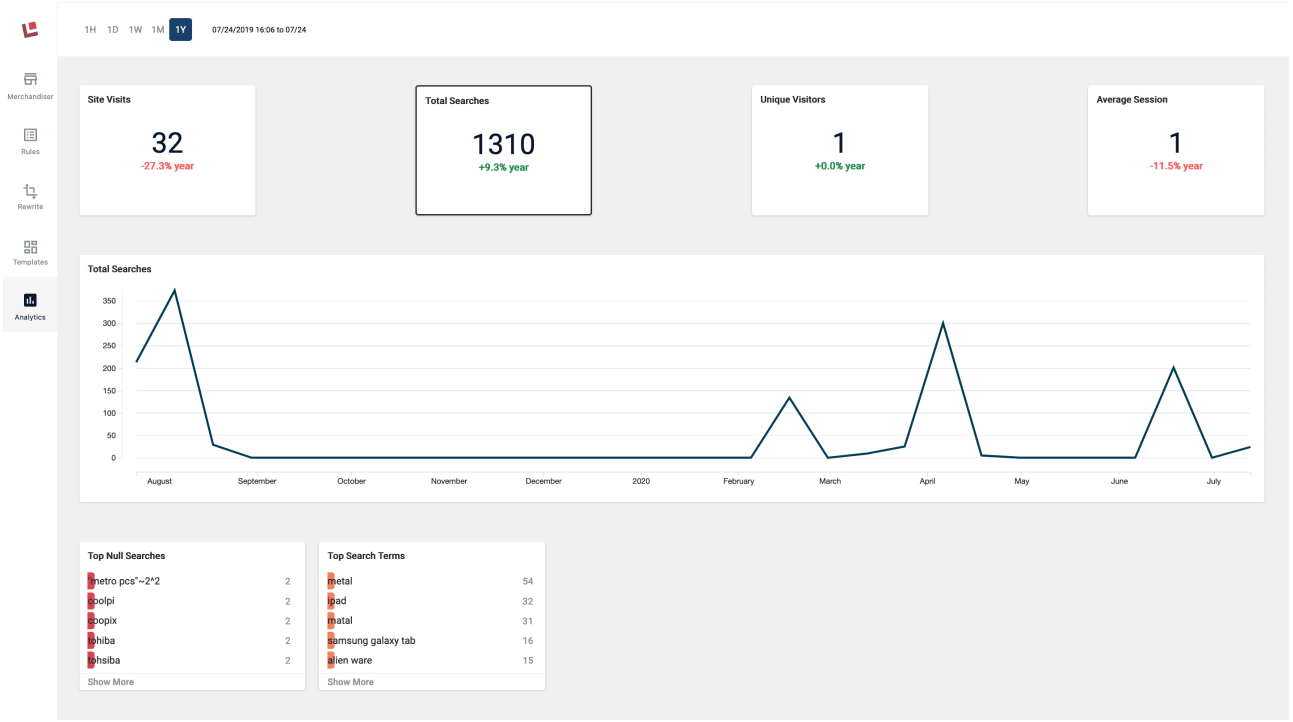


You can edit the query rewrite rule by hovering over this indicator and clicking the edit button. A dialog box will appear:



Similarly, when a business rule is triggered by a search condition, it will appear in the right panel of the Simulator:

Analytics



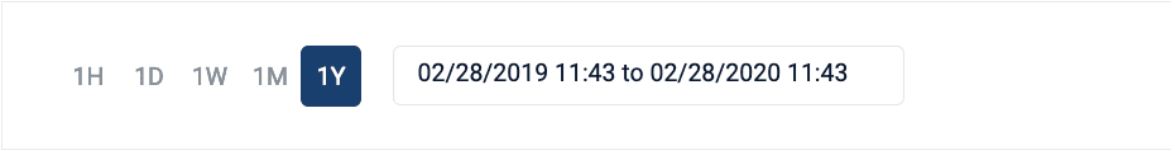
The Analytics screen is available as part of the *Predictive Merchandiser*, a separate offering from Fusion. See the *Predictive Merchandiser documentation* for more information.

Learn how to use the *Rules Editor Analytics screen*.

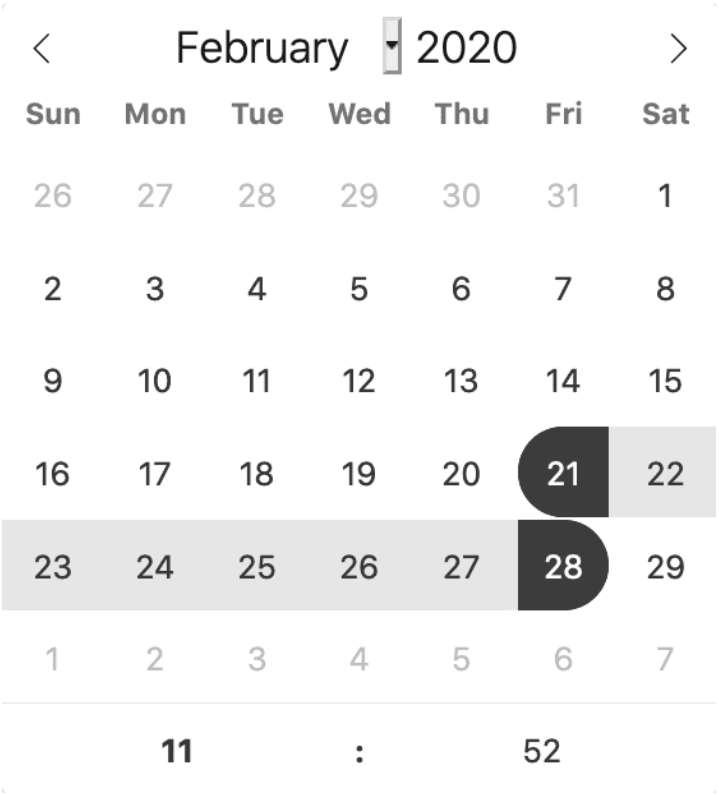
UI elements

Data date range

- **Quick Range Selectors.** Click one of the quick range selectors to facet your data.



- **Calendar Selector.** Use the calendar selector to specify a start and end date and time facet for your data.

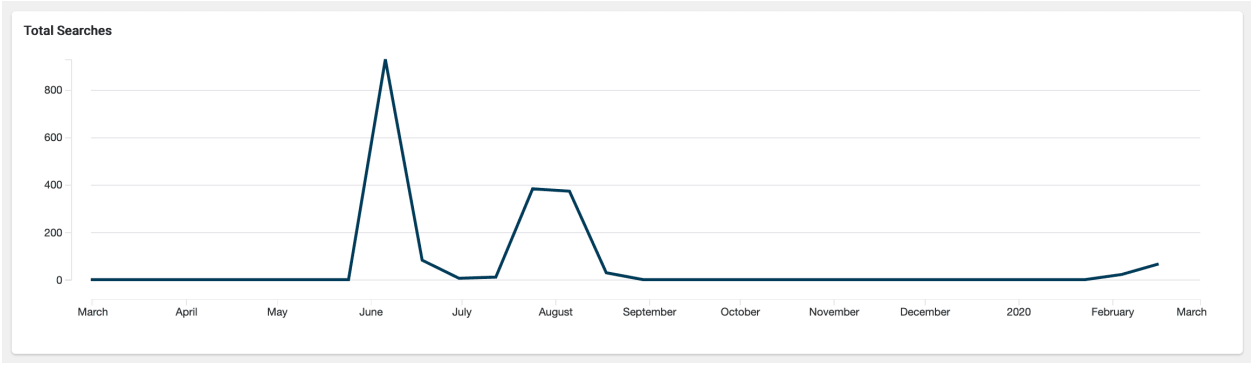


Visitor data

- **Data Cards**

  - **Site Visits.** The total number of visits.
  - **Total Searches.** The total number of searches performed.
  - **Unique Visitors.** The number of unique visitors to your site.
  - **Average Session.** The average length of time spent on your site.

- **Data Graph.** Detailed information for the selected data card.



Stopwords Files

Fusion collections are Solr collections that are managed by Fusion. Solr itself manages a set of resources *(https://cwiki.apache.org/confluence/display/solr/Managed+Resources)* for a collection.

Stopword lists are one such resource.

Stopwords are words that are filtered out of a user-entered search request. Examples of common English stopwords are *a*, *an*, and *the*.

The Fusion UI provides a stopwords manager tool that is reached from the **Stopwords** page in the **Collections** menu. This tool lets you view the contents of all configured stopwords files in an editable browser.

Curating Your Search Experience

Predictive Merchandiser and Experience Optimizer are both AI-powered tools that provide insights, customer recommendations, and search optimization in a single comprehensive product.

Determining the best product for your needs depends on the type of materials you would like to tailor your search experience for.

Experience Optimizer vs Predictive Merchandiser

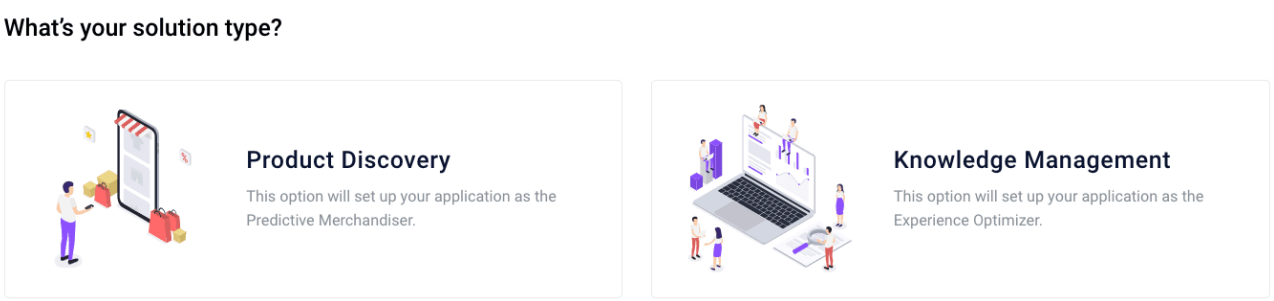
Experience Optimizer is typically for Knowledge Management while Predictive Merchandiser is typically for eCommerce. Knowledge Management uses files and documents and wants to see additional text and links, with less emphasis on images. Online stores tend to put a greater emphasis on images than text.

In both situations, items chosen as most important are put into zones and shown for all users with natural search results below. You can pin important products, files, or folders as the first links.

Product	Experience Optimizer	Predictive Merchandiser
Layout	Optimizes list view	Thumbnail based grid view
Images	Small thumbnail; if no image available then show generic images	Larger images
Description	Page title with heading and excerpt	Less text, typically just a title

Choosing the right product for your needs

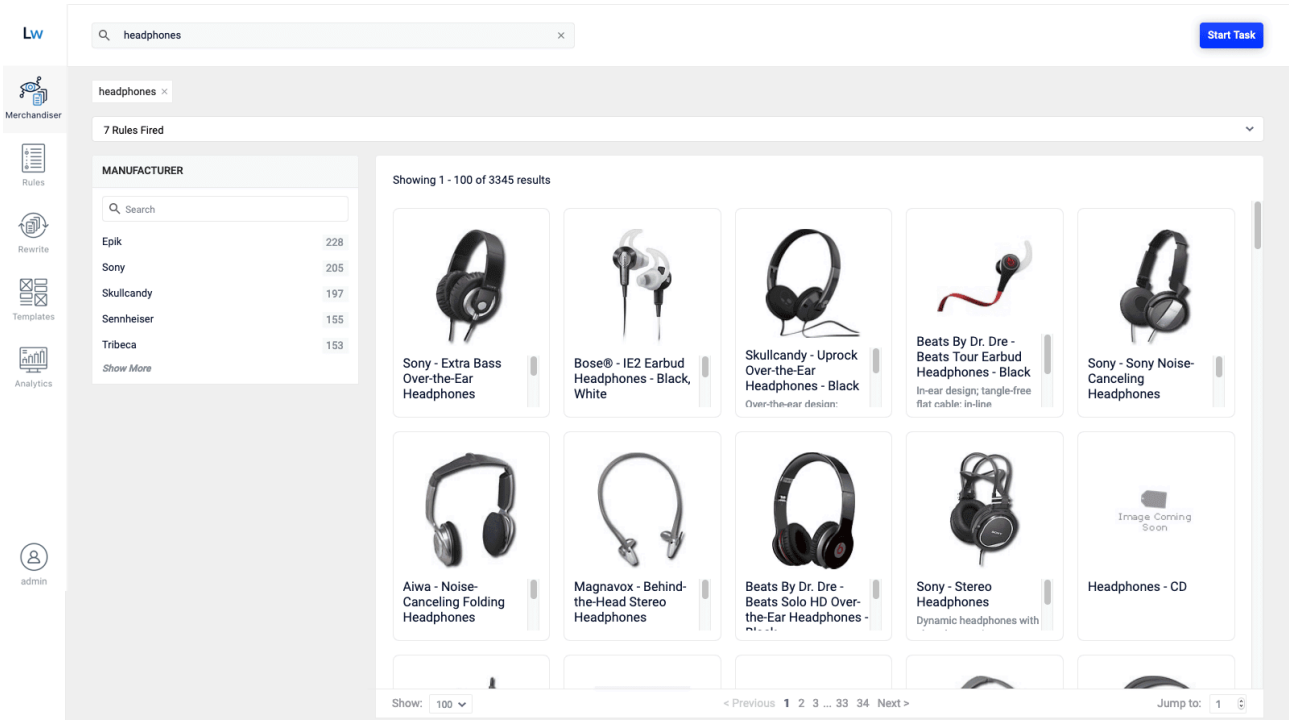
When you enter the **Rules** section from Fusion for the first time, you will initiate a wizard that will allow you to choose between two solution types, **Product Discovery** or **Knowledge Management**. From there, you will see your prospective workspace with predefined defaults.



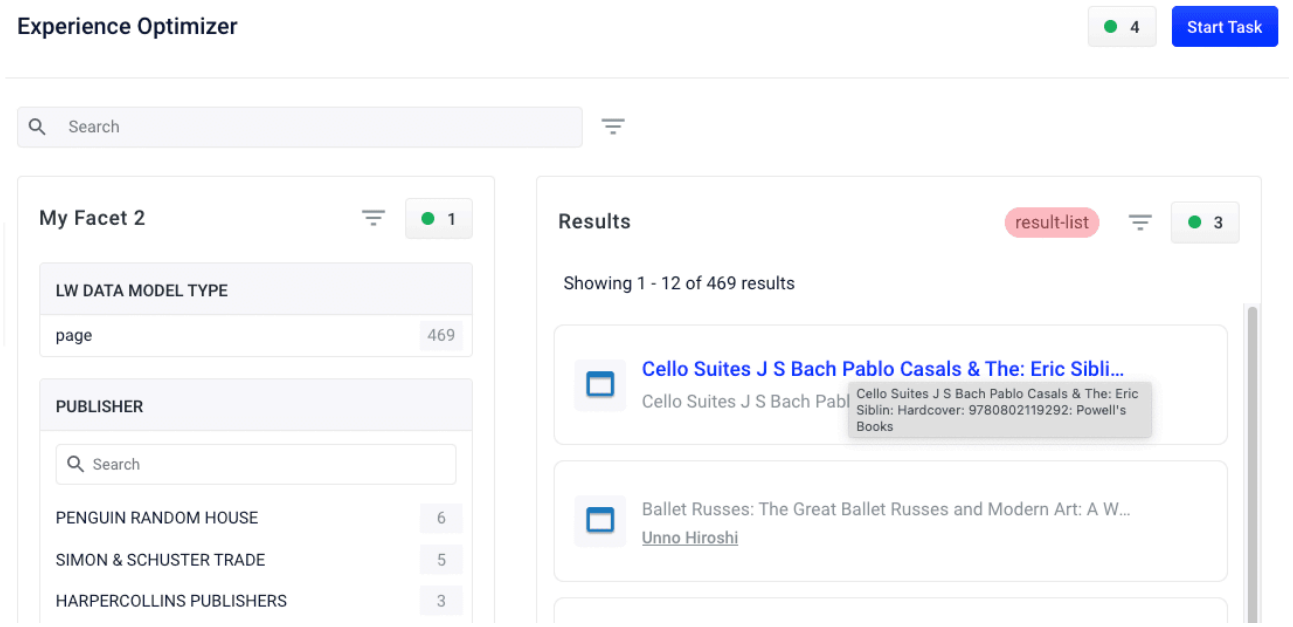
This selection is permanent. Please select with caution.

Exploring solution types

The Product Discovery route activates Predictive Merchandiser. This route is designed for online retailers and similar clients who want to create an enhanced shopping experience for their customers. *Predictive Merchandiser's* grid-view interface is helpful for displaying products in the best light, with larger product imagery and a product title.

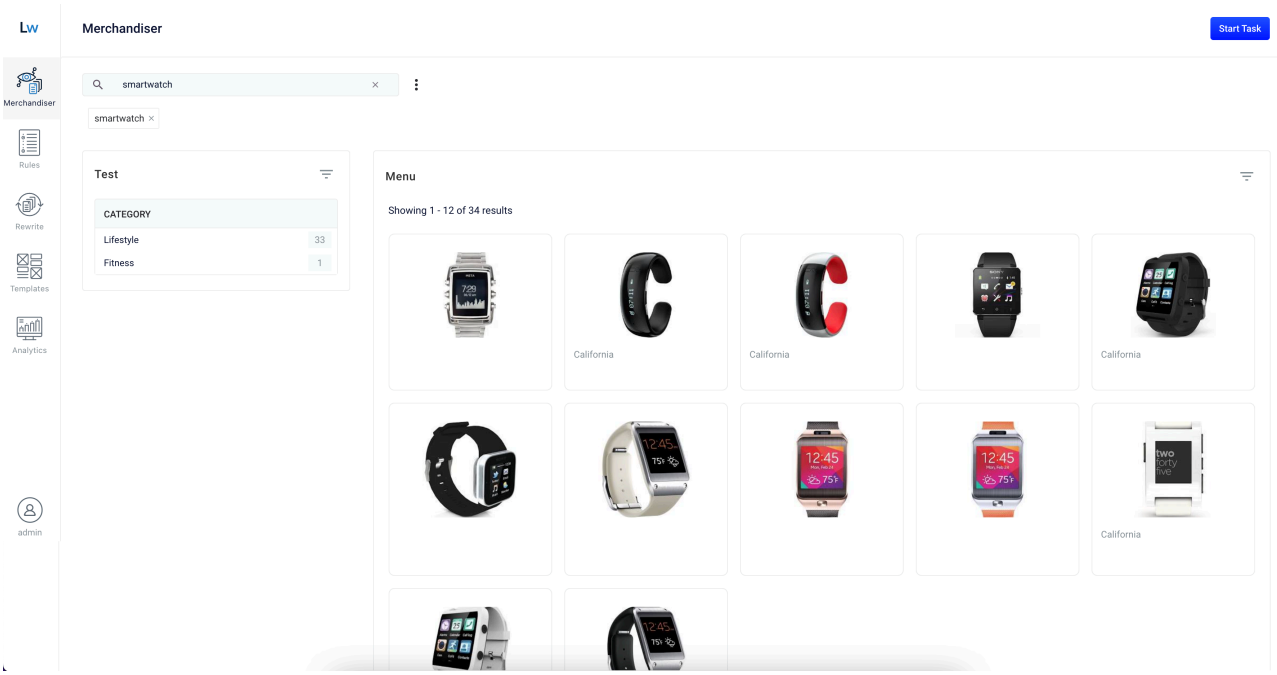


The Knowledge Management route activates Experience Optimizer. This route is intended for search managers and similar clients who want to facilitate access to documents, files, or important news. *Experience Optimizer's* list-view conveys more information in search results with additional lines of text and smaller imagery. It also contains assigned imagery for abstract items, or items with no image.



Predictive Merchandiser

Predictive Merchandiser is an AI-powered merchandising tool for Product Discovery solutions. It provides insights and recommendations for optimizing search results and product placement.



It provides an easy to use interface, allowing you to:

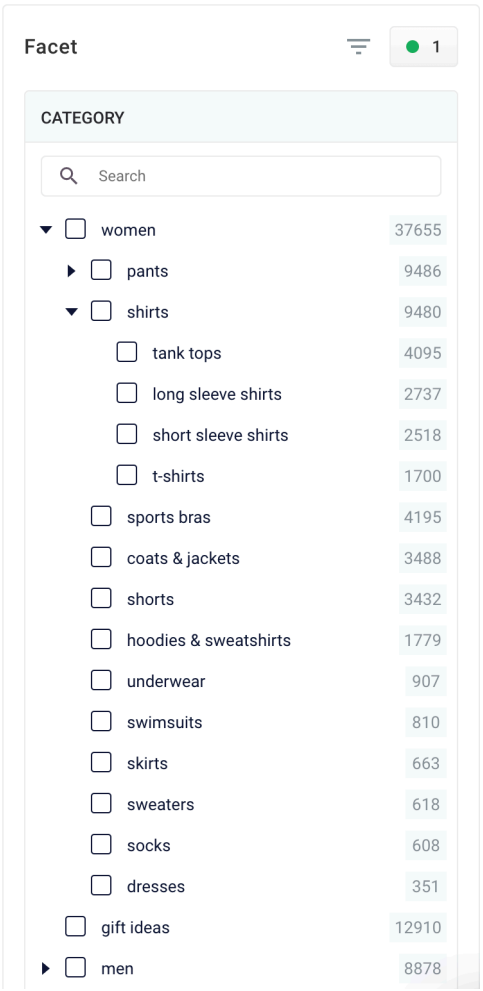
- Connect to a Fusion data source and simulate the shopping experience of your customers.
- Pin, boost, bury, and block specific products in your catalog.
- Create business rules to display custom banners, or redirect customers to a different page on your site, for example.
- Create search rewriting strategies to manage spelling mistakes, synonym matching, phrase matching, and poorly performing queries.
- Review query rewriting strategies that were generated automatically using machine learning models in Fusion.
- Deploy new business rules and search rewrites to the Fusion data source where they can be triggered by live queries.
- View analytics reports that show, for example, the variation in site visits over time, the most clicked on products, and the most searched for terms.

Additional resources

- View our training on Predictive Merchandiser: Getting Started *(https://academy.lucidworks.com/predictive-merchandiser-getting-started-with-predictive-merchandiser)*.

Hierarchical Facets

Facets of multiple levels under a parent facet are known as hierarchical facets. Hierarchical facets allow faster navigation through facet submenus and breadcrumbs when navigating through the available options. Facets with children can be indicated by an arrow to indicate they have children that become visible when the arrow is selected. Selecting a child facet after expanding the facet menu allows the user to bypass the options between the top page and their choice.



Hierarchical facets are split according to the delimiter set in the *Apply Rules* query pipeline stage.

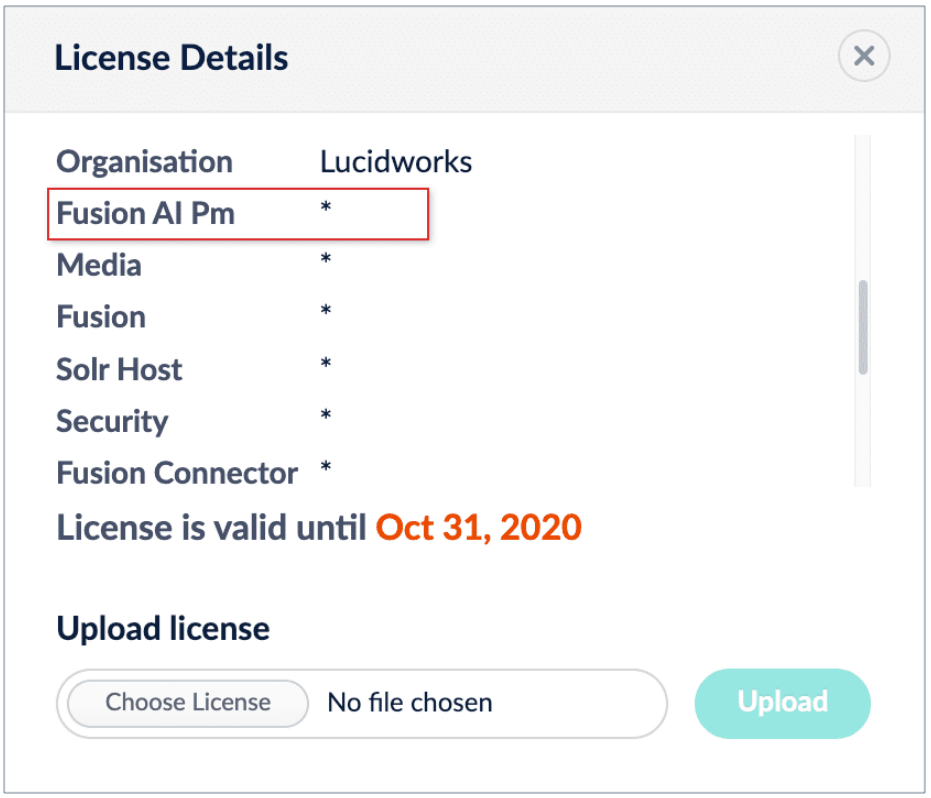
For more information, see *How to Use Hierarchical Facets*.

Licensing for Predictive Merchandiser

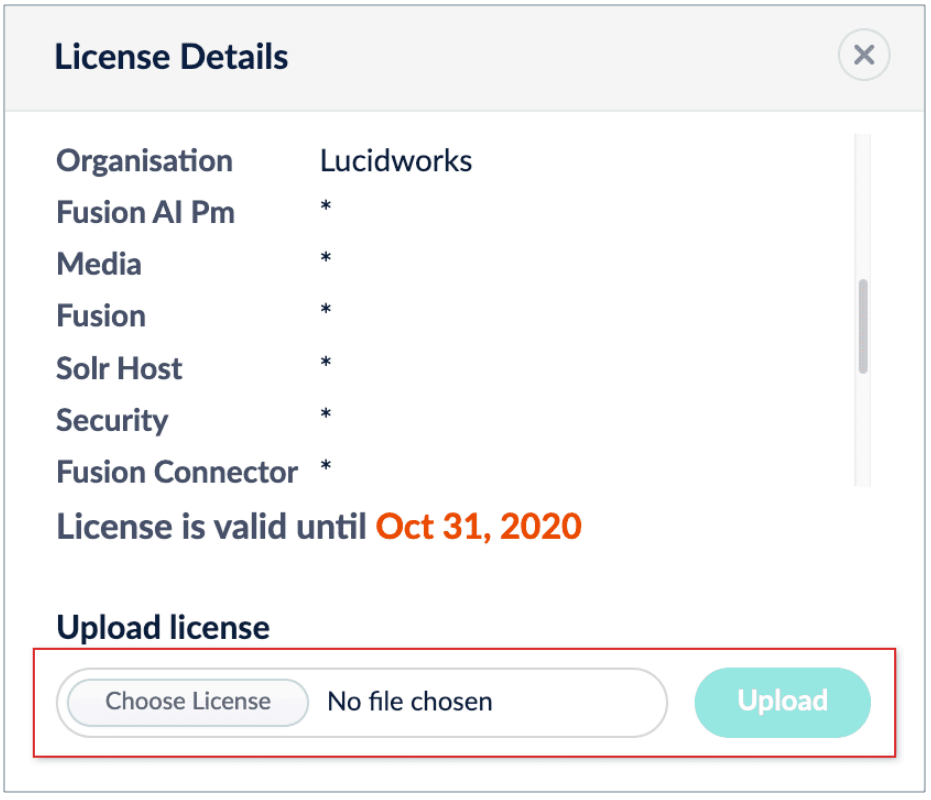
Predictive Merchandiser is integrated with Fusion and requires a valid license.

To verify your Predictive Merchandiser license, click the **Account Settings** icon and select **License Details**.

If you have a valid Predictive Merchandiser license, a * displays after **Fusion AI Pm**:



If you do not see a *, you must upload a valid license file to use Predictive Merchandiser:



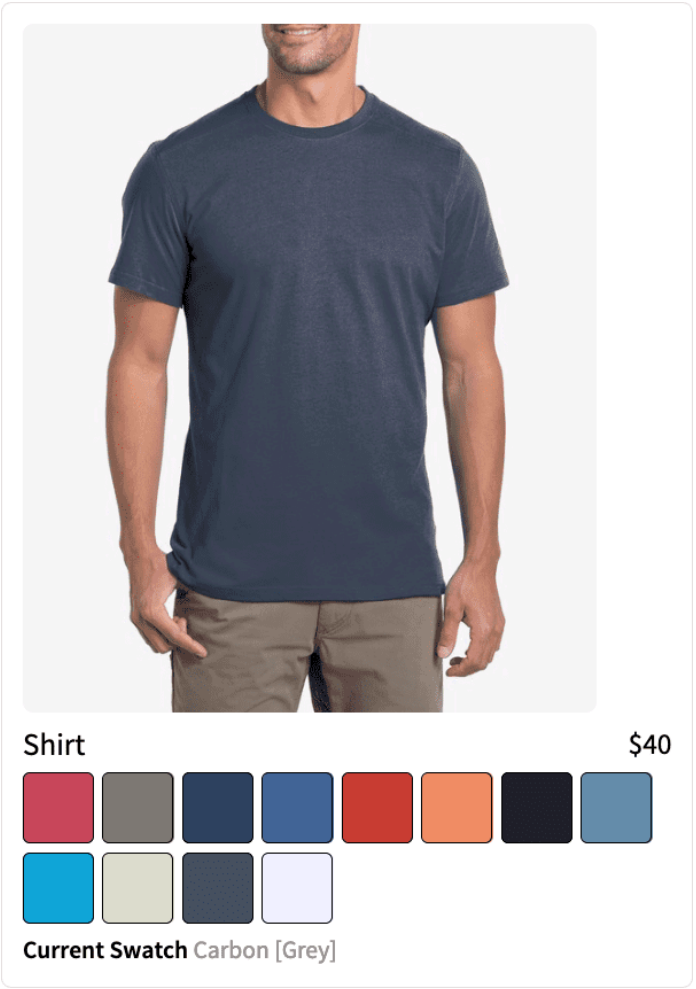
Merchandising Zero Results

As a merchandiser, search queries that generate zero results can drive customers away from the site. Predictive Merchandiser allows you to ensure this does not happen. Instead of display zero results, you can choose to display certain products when a customer reaches a zero results page.

Learn how to use the *merchandising zero results* feature.

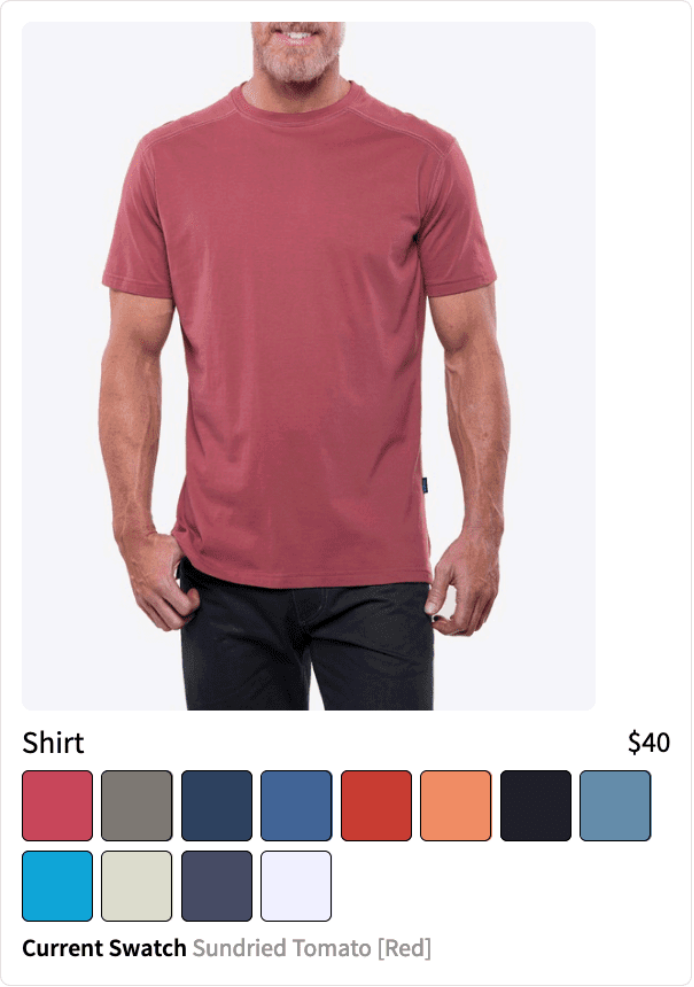
Product Grouping

Predictive Merchandiser makes it easy to group products together so they can be presented to the user. This can be useful when you want to show a product in all of its color variants or the various versions of a product (basic or advanced, for example).



Product groups can be boosted or blocked. If you want to promote a product group that is on sale, for example, you can boost the group to the top of the results.

You can still apply rules individually to products within a group. This allows you to boost a product based on a specific user's search behavior. If a user typically favors the color red, you can boost the red variant of the product to the top of the group.



Best practices

These parameters group products, and are defined in the Additional Query Parameters Stage:

Parameter Name	Parameter Value	Update Policy
group	true	default
group.format	grouped	default
group.ngroups	true	default
group.field	style_id_s	default NOTE: You must configure `groupField=<field>` if you use *App Studio* to create your search application.

For more information about adding a query pipeline and additional query parameters, see:

- *Add a Query Pipeline Stage*
- *Additional Query Parameters Stage*

Collapse/Expand parser grouping

The Collapse/Expand parser allows grouping with the following parameters:

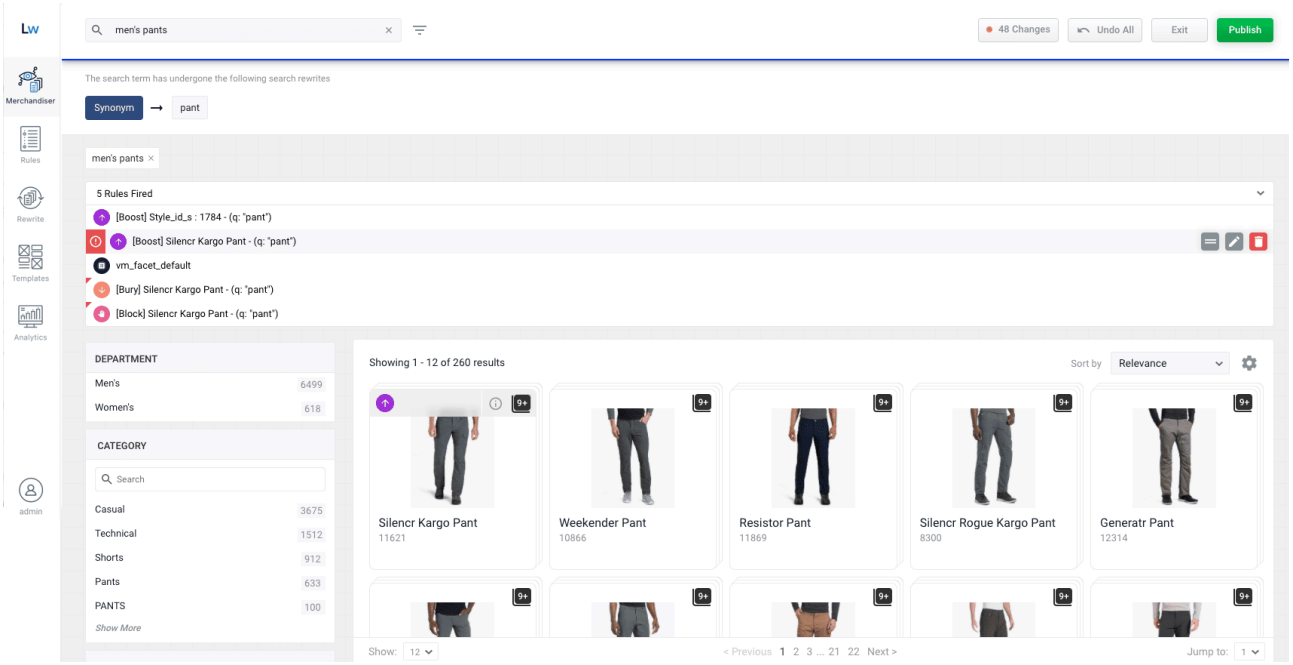
Parameter Name	Parameter Value
expand	true
enableElevation	true
group	false

See Collapse and Expand Results *(https://solr.apache.org/guide/collapse-and-expand-results.html)* for more information.

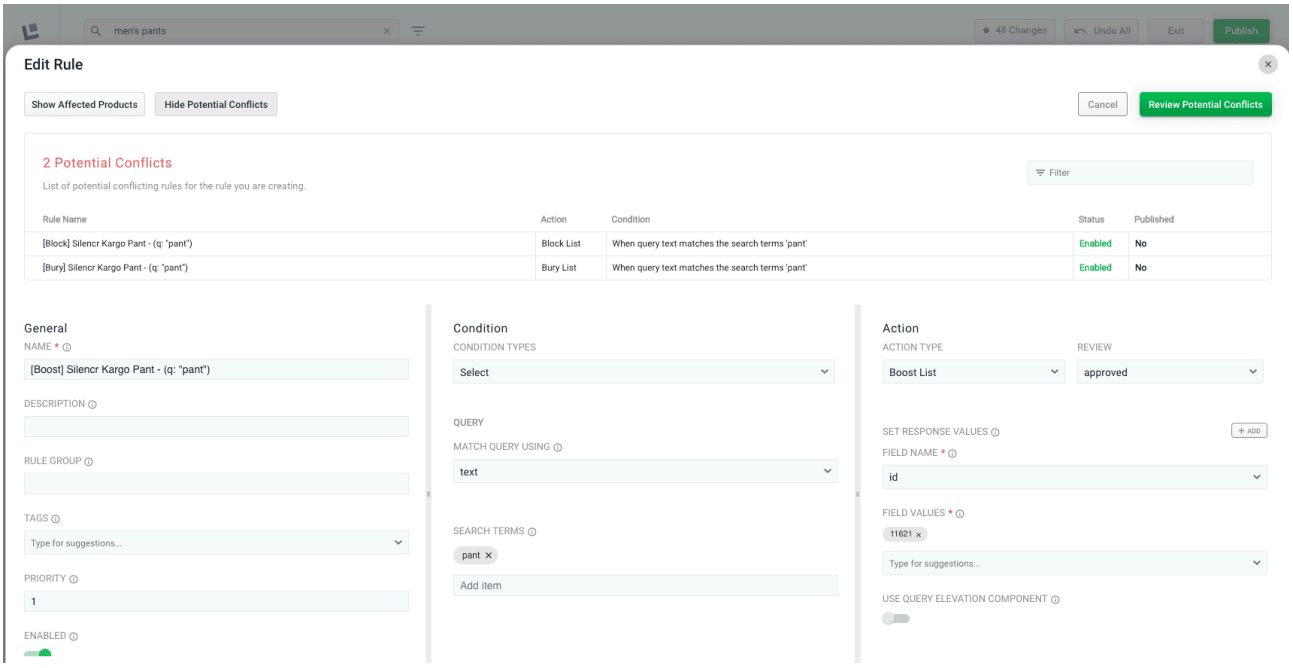
Rule Conflict Resolution

Sometimes, rules created in the *Rules Editor* and Predictive Merchandiser conflict with each other. For example, if you have three rules to boost, bury, and block the same product, Predictive Merchandiser highlights these conflicting rules so you can address them.

To highlight conflicting rules, the **Rules Fired** section displays a red warning icon next to the dominant rule and a red flag on the conflicting rules.

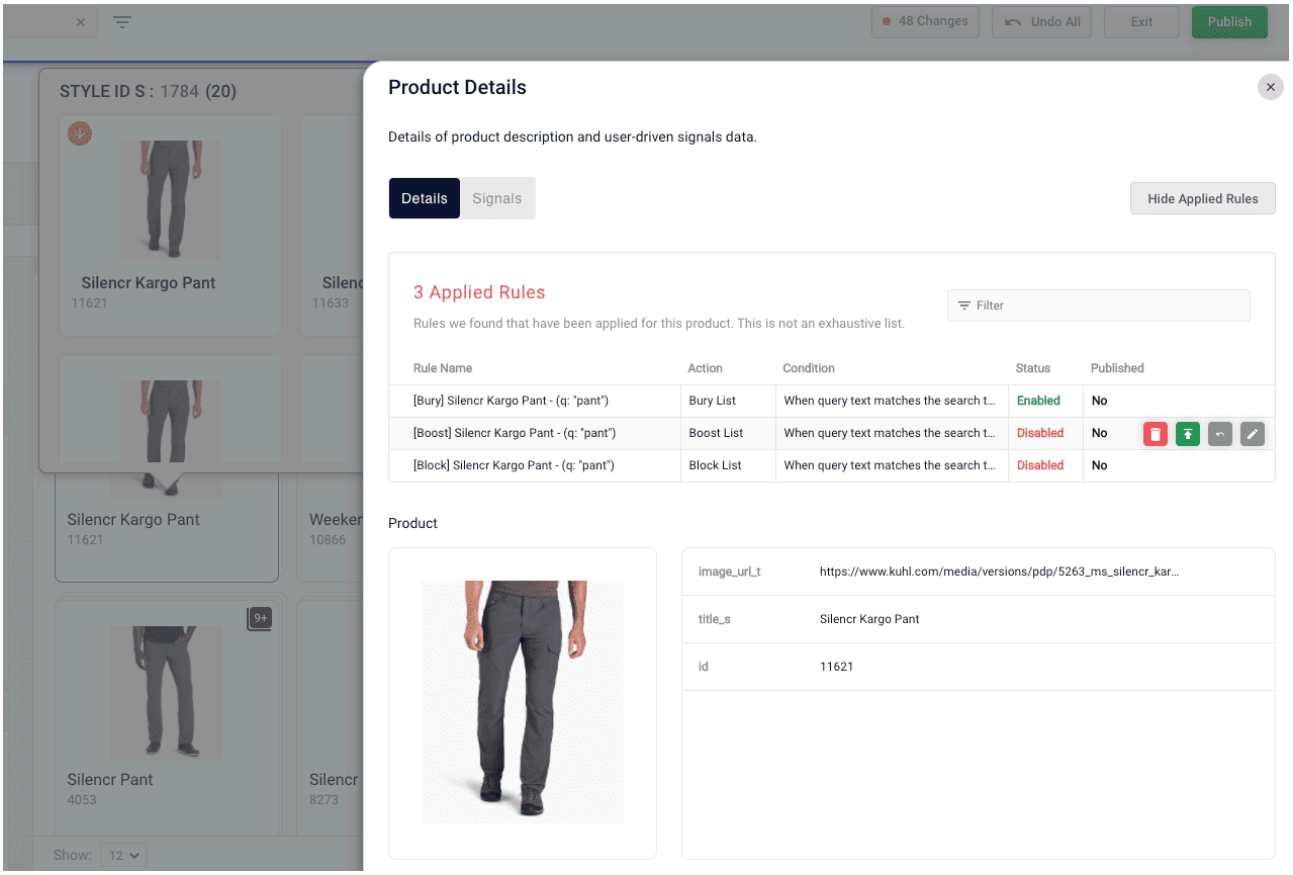


Click the **Edit** button to view the details of the conflicting rules and resolve the conflict.



The details of the conflicting rules affecting specific products are found in the details panel. Click the **Start Task** button

, hover over a product, and click the **Details** button .



Search Rewrites

Search rewriting is a way of modifying search queries so that they more accurately reflect the intentions of your customers. You can create a misspelling search rewrite, for example, to correct a spelling mistake that commonly occurs when searching for a particular product. The misspelled query will then produce the same results as a correctly spelled search.

There are four rewrite types:

- *Head/Tail* - improves poorly performing searches.
- *Misspelling detection* - corrects common spelling mistakes.
- *Phrase detection* - identifies products with matching phrases.
- *Synonym detection* - includes alternative words with the same meaning.

In Predictive Merchandiser you can create search rewrites manually. You can also edit, test, review and publish the search rewrites that are generated automatically in Fusion from signals data.

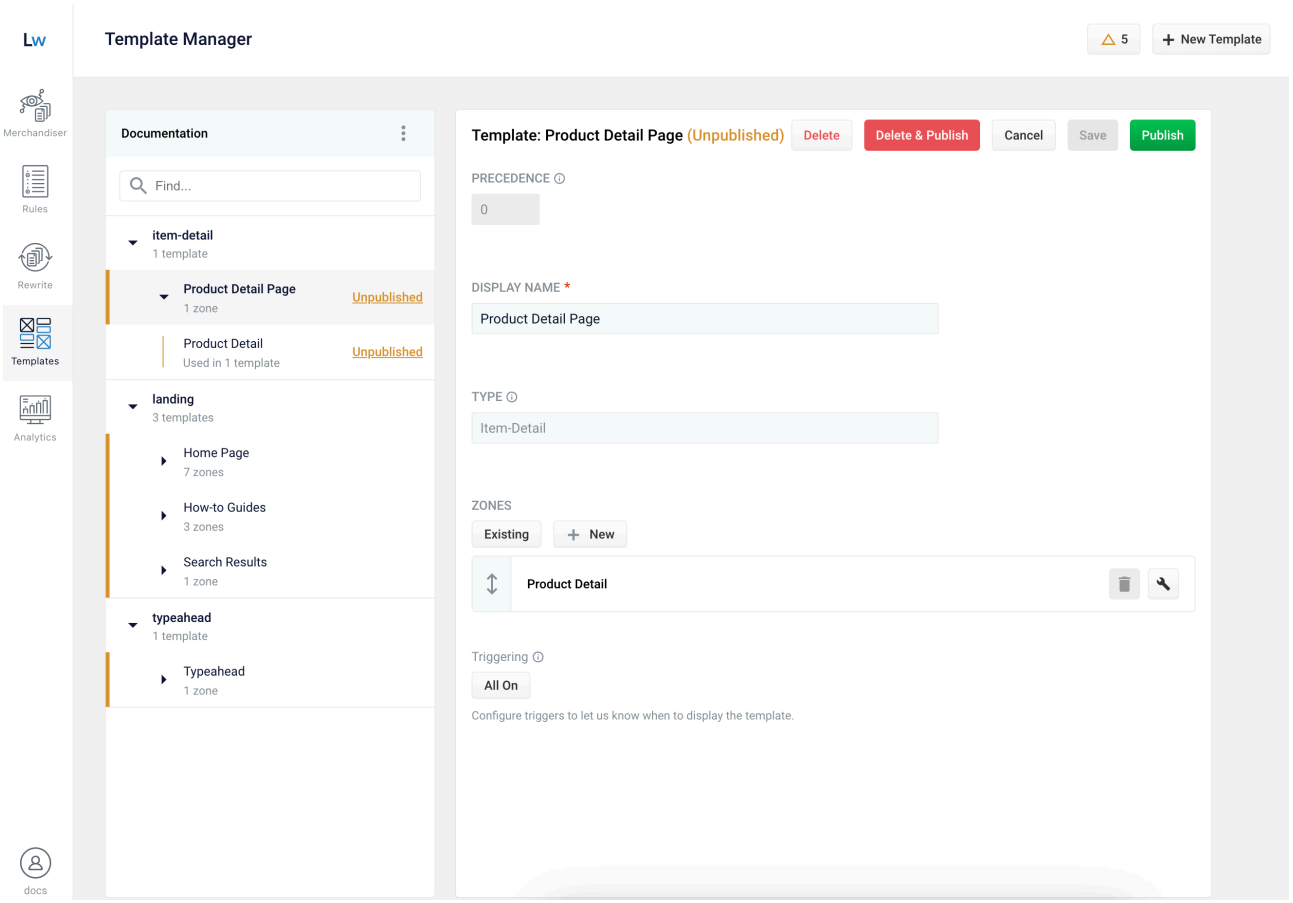
Supported Browsers

Currently, Predictive Merchandiser is officially supported in Google Chrome *only*. Although Predictive Merchandiser is available in unsupported browsers, features and functionality may be limited.

Browser	Supported
Google Chrome	✅
Firefox	×
Safari	×
Opera	×
Edge	×

Templates

The Templates screen in the Rules Editor allows you to design, test, and implement a wide variety of search experiences. Results are conditionally delivered from multiple different query pipelines, enabling you to use Fusion's search, browse, and AI-driven functionality throughout your site.



Access Templates from the *Rules Editor* by navigating to **Relevance > Rules > Templates**. See *Use Predictive Merchandiser Templates* for instructions on using Templates.

Templates are only available with a valid license for *Predictive Merchandiser*. If available, you will see the templates icon in the left navigation panel:

For configuration information, see *Templates UI Configurations Reference* and the *Templating API Configuration Reference*.

Templates

Templates are comprised of two primary elements, the *template* itself and its associated *zones*. A variety of conditions, or *triggers*, determine which template is shown to the customer. These triggers can include whether the template is active, the template's precedence over other templates, the customer's query, and whether the template is triggered within a specific date range (if set).

Default templates

A new app starts with built-in templates: Detail Page, Search Results, and Typeahead. The Detail Page and Search Results Page templates are immediately available in the **Templates** screen.

Detail Page
: The Detail Page defines what a customer sees when they view the details of a product. It can be configured in the **Templates** screen or by hovering over a product in the **Predictive Merchandiser** screen and clicking the **Detail Page** button that appears.

Search Results page
: The Search Results Page defines what a customer sees when they reach a search results page. This template can also be configured in the **Template** screen. Alternatively, you can click **Merchandiser > Start Task > Edit Template**.

Typeahead Template

The Typeahead Template allows you to configure what is displayed when query terms are typed into the search bar. To add this template, click **Merchandiser > Start Task > enter a search query term**. When the system displays "Typeahead not created. Do you want to create a new typeahead template?", click **New** to create the typeahead template.

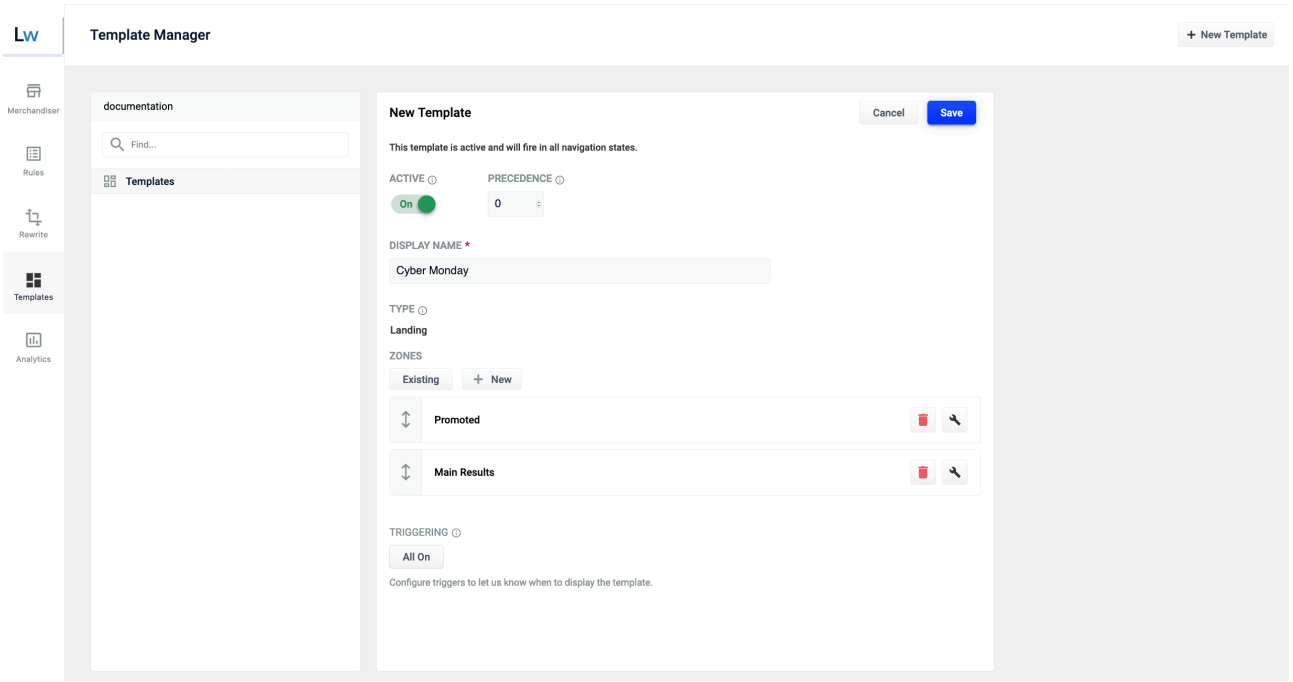
Custom templates

In some cases, you may want to create custom template types to add more options for curating pages in your search application. You can create a custom template type in the **Template** screen by following the steps in *Create a Custom Predictive Merchandiser Template Type*.

Zones

Zones enable Fusion users to create rules designed for specific query profiles. Combined with templates, this allows a search application to display results from multiple query profiles in the same view.

For example, if a customer searches for `climbing`, one zone can fetch results for climbing apparel, and another zone can fetch blog posts about climbing. The same zone can be used in multiple templates, and changes to a zone will be made on all templates that zone is used in.



In the example above, a unique template is fired for the query `cyber monday`. This template has two zones:

- 1 **Promoted** - Display advertised products.
- 2 **Main Results List** - Display general search results.

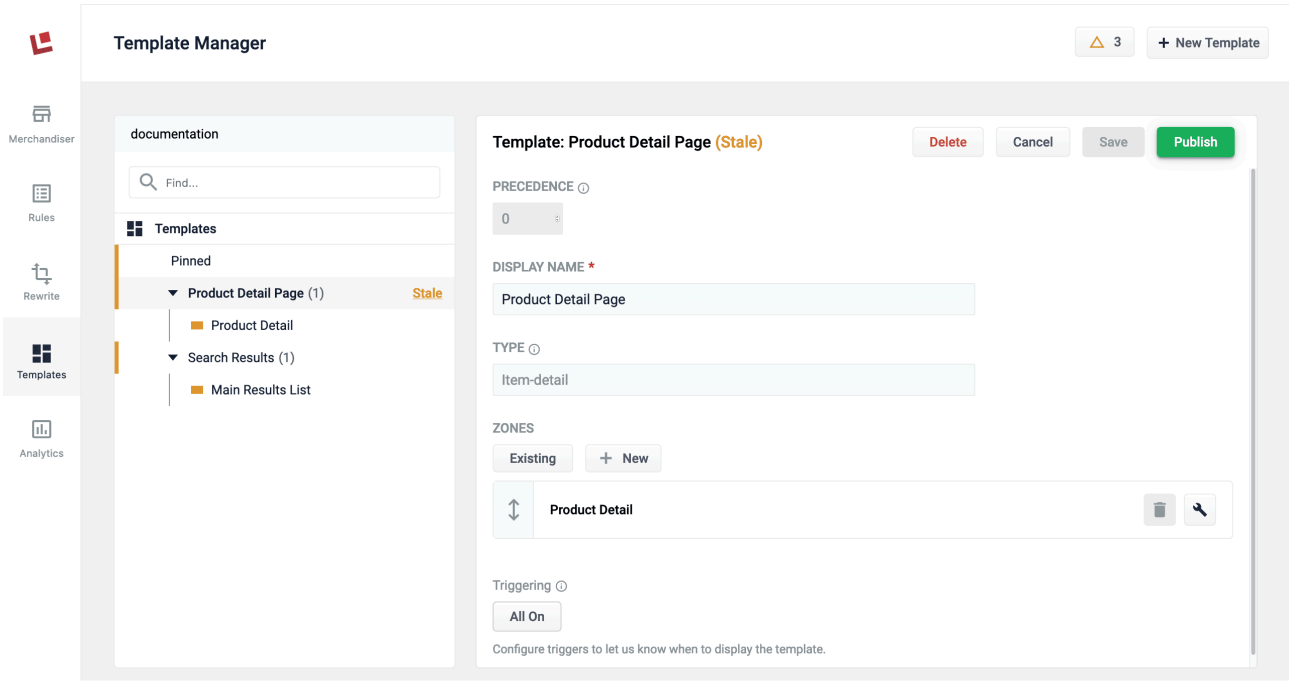
Templates UI Configuration Options

The Templates screen in the Rules Editor allows you to design, test, and implement a wide variety of search experiences. Results are conditionally delivered from multiple different query pipelines, enabling you to use Fusion's search, browse, and AI-driven functionality throughout your site.

Configuration options

Templates

When creating or editing a template, the status, such as active or stale, is indicated after the template name.



The active status can hold the following values:

Tag	New	Saved	Published	Active
Editing	✅			
Stale		✅		
(nothing)			✅	✅

Configuration options for templates:

Option	Description
Precedence	If multiple templates are triggered, this setting determines which template is displayed. Lower numbers are considered higher precedence. For example, a template with precedence of `999` will trigger before a template with precedence of `1`.
Display Name	The name of the template.
Type	The type of template. This is an arbitrary value classifying the template. It has no effect on triggering/rendering logic unless the value is `Typeahead`. If `Typeahead` is assigned, the template will be excluded from `/trigger` and `/render` endpoints. This value is read-only and for informational purposes only.
Zones	Determines which zones are loaded when the template is triggered. Zones are displayed in the order they appear.

Triggering	The conditions that must be met for the template to display.
	
	• **Date Range.** Define a date range to trigger the display of the template. For example, a desired template may be set to display during a Black Friday sales event.
	• **Trigger.** Define triggers to trigger the display of a template. Triggers are added with the **Trigger** button.
	  ◦ **Search Term.** Search terms required in the query for this trigger to match. The configured analysis will be applied to tokenize and process these terms.
	  ◦ **Match Mode.** Defines search terms matching criteria.
	  ▪ `phrase` requires the search tokens of this trigger to exist as a phrase within the search terms of the query. The phrase must be in the same order.
	  ▪ `all` requires the search tokens of this trigger to exist within the search terms of the query in any order.
	  ▪ `exact` requires that the search tokens in this trigger exactly match the search tokens in the query *in the same order*. As with all match modes, `exact` still applies the configured analysis, such as stemming. Singular and plural forms will be considered identical despite not being *exactly* the same.
	  ◦ **Filters.** Filters (`fq`) values required in the query for this trigger to match. Additional filters may be present in the query and still satisfy this trigger.
	  ◦ **Exact.** If **On**, the nav state must match exactly this trigger's criteria. For example, no extra filters are allowed, and no search terms are allowed if this trigger has none defined. Not to be confused with setting **Match Mode** to `exact`, which specifically defines the search term matching logic.

Matches All Locations	If set to **On**, this template is always displayed, and triggers are ignored.

Triggering

The date range trigger is activated alongside triggers with an `AND` logical operator. Conversely, triggers are activated alongside other triggers with an `OR` logical operator. For example, consider the following scenarios:

Scenario 1		Scenario 2	
Date Range	*none*	Date Range	`November 25, 2020 – December 5, 2020`
Search Term	`cyber monday`	Search Term	`cyber monday`
Search Term	`sales event`	Search Term	`sales event`

In **Scenario 1**, no date range is configured. The template is triggered if:

- The user searches the term `cyber monday` *or* `sales event`.

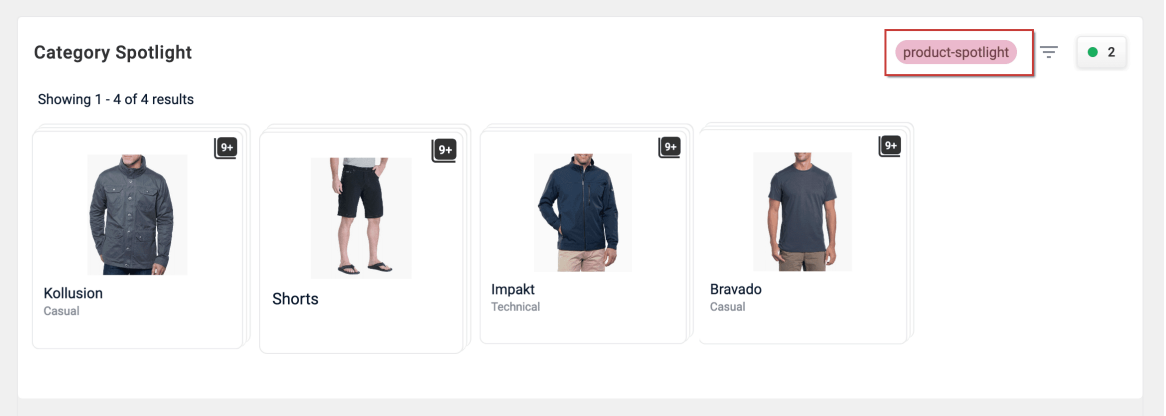
In **Scenario 2**, a date range is configured. The template is triggered if:

- Today's date is within the date range `November 25, 2020 – December 5, 2020`, *and* the user searches the term `cyber monday`
- Today's date is within the date range `November 25, 2020 – December 5, 2020`, *and* the user searches the term `sales event`

The template is *not* triggered if:

- Today's date is not within the date range `November 25, 2020 – December 5, 2020`
- Today's date is within the date range `November 25, 2020 – December 5, 2020`, but the user does not search the term `cyber monday` *or* `sales event`

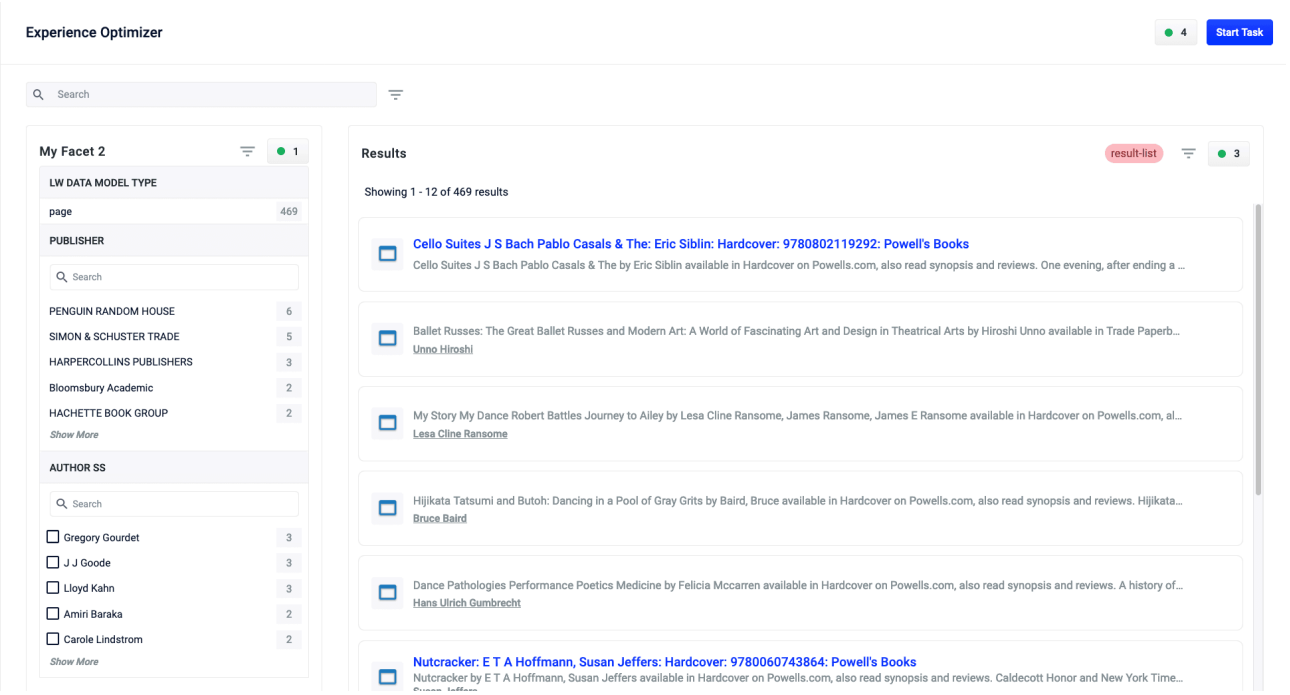
Zones

Option	Description
Display Name	The name of the zone.
Type	An arbitrary value classifying the zone. It has no effect on triggering/rendering logic. Intended only to be used by the PM UI. This value is read-only and for informational purposes only.
Query Profile	Determines query profile the zone will fetch results from.
Tags	Tags are passed to the query profile when rendering. They are combined with tags found in the request parameters of the render request.
Omit Filters from Query	If set to **On**, the filters (`fq`) are ignored. For example, if you want the zone to display items with type `sunglasses` even though the user has filtered the results to `shorts`, enabling this setting allows the zone to display sunglasses.
UI Treatment	Arbitrary text that is included in the response when rendering the zone. This is generally used by the frontend to determine how to display the zone. UI Treatment tags accept a color value to help distinguish them in the Predictive Merchandiser screen: 
Results Layout	Select grid or list view.
Display Fields	Add field names and how they are displayed (type of field).
Product Image	Field name for the product image.
Product Title	Field name for the product title.
Product Description	Field name for the product description.

Product Image, Product Title, and Product Description are only used for rendering products in Predictive Merchandiser. Your search application will determine what fields are used in production.

Experience Optimizer

Experience Optimizer is an AI-powered search tool that optimizes results for enterprise assets.



Use cases

Reduce administration overhead

Experience Optimizer is a way for internal admin to build out pages necessary to find information such as benefits enrollment, hot links for new hires, or topics/docs other employees are looking for. At benefits enrollment time, HR can take over the top zone of the page to reduce repeated requests about links, where to go, and what to do, thus saving time and resources. All such zones and templates can be built without the need of a developer.

Content importance

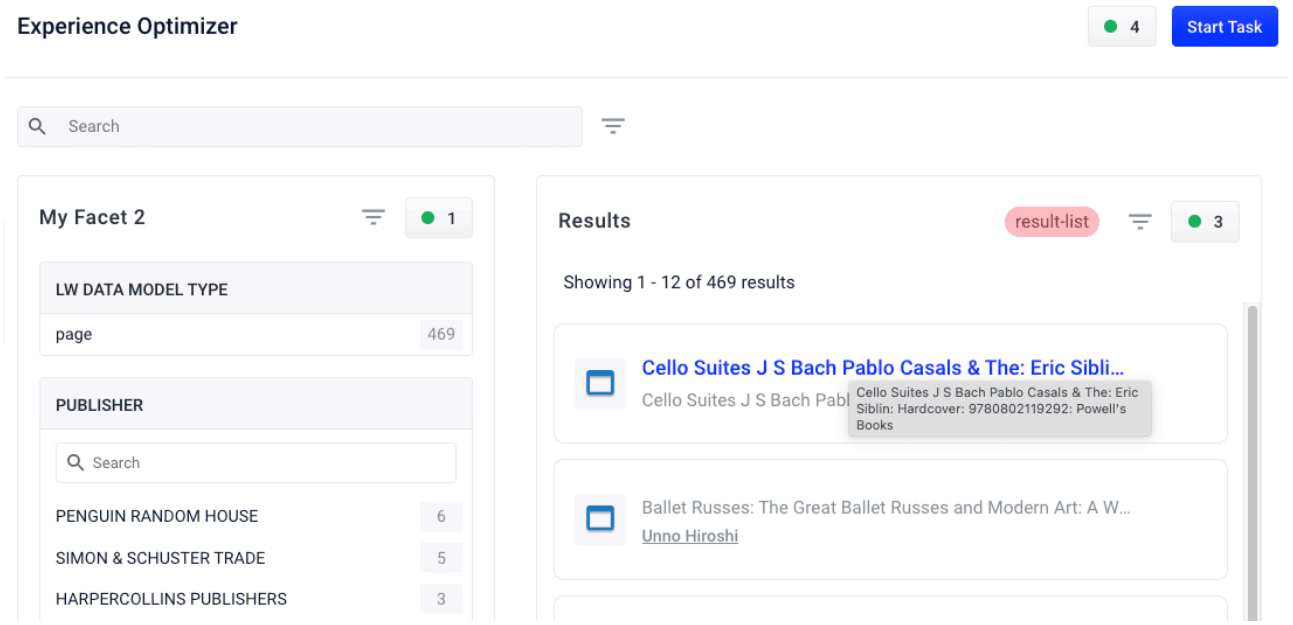
You can build a zone for blog posts at the bottom of a page and set a pipeline to only pull in blog posts and sort by date.

Trending articles

Use signals or click throughs to show trending content for all users. Trending articles shows what's important while also phasing out information that becomes irrelevant over time.

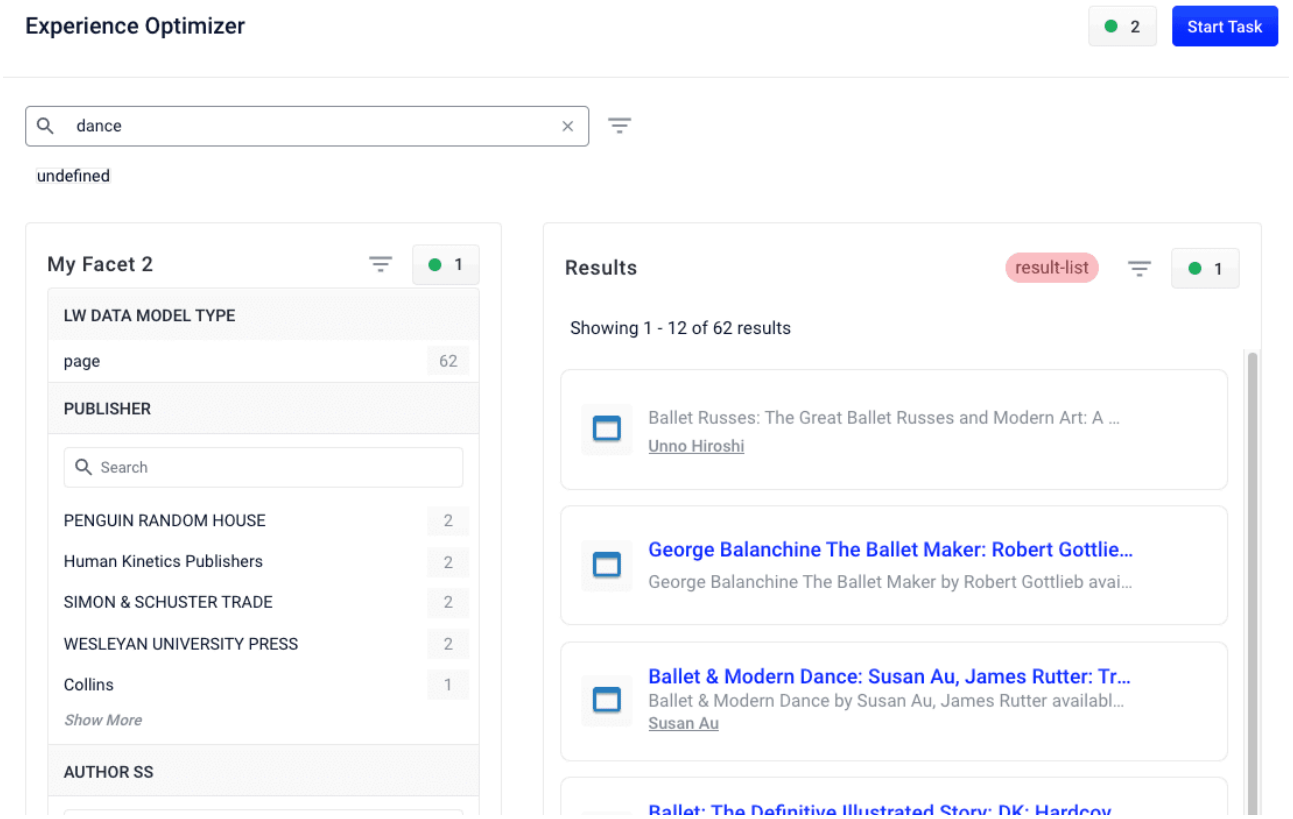
Document Grouping

Experience Optimizer makes it easy to group documents together so they can be presented to the user. This can be useful when you want to show several documents that are relevant to each other, for example.



Document groups can be boosted or blocked. If you want to promote a document group that is new, for example, you can boost the group to the top of the results.

You can still apply rules individually to documents within a group. This allows you to boost a document based on a specific user's search behavior. If a user that is typically interested in ballet searches for "dance" content, you can boost ballet-related articles to the top of the group.



Best practices

These parameters group documents, and are defined in the Additional Query Parameters Stage:

Parameter Name	Parameter Value	Update Policy
group	true	default
group.format	grouped	default
group.ngroups	true	default
group.field	style_id_s	default NOTE: You must configure `groupField=<field>` if you use *App Studio* to create your search application.

For more information about adding a query pipeline and additional query parameters, see:

- *Add a Query Pipeline Stage*
- *Additional Query Parameters Stage*

Collapse/Expand parser grouping

The Collapse/Expand parser allows grouping with the following parameters:

Parameter Name	Parameter Value
expand	true
enableElevation	true
group	false

See *Collapse and Expand Results* for more information.

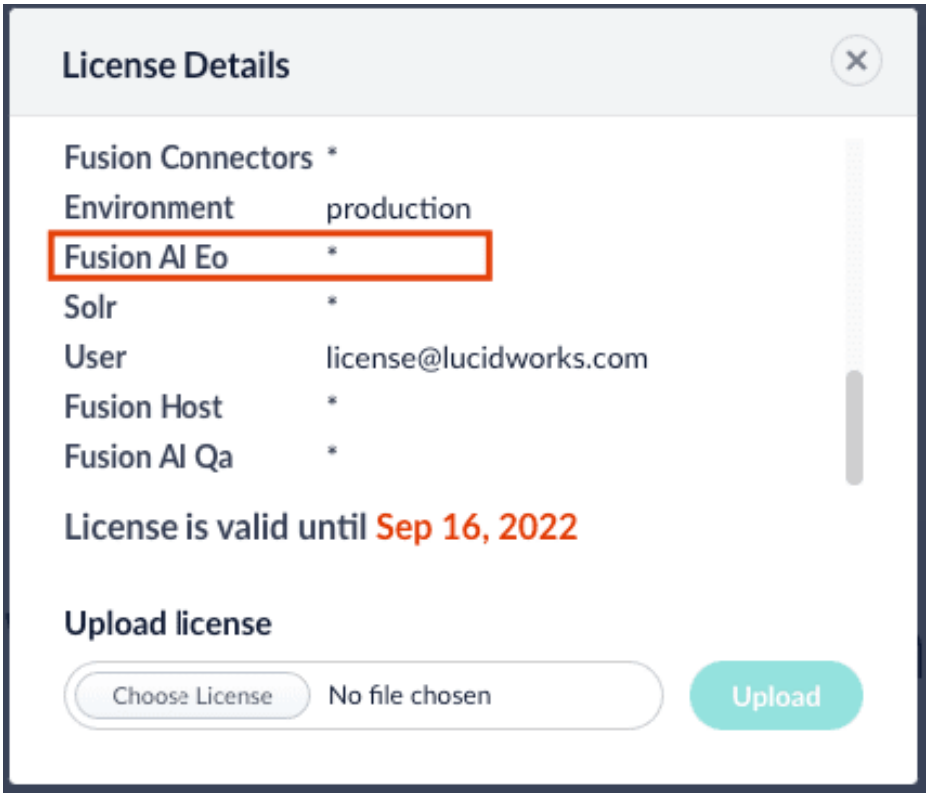
Licensing for Experience Optimizer

Experience Optimizer is a Fusion feature and requires a valid license.

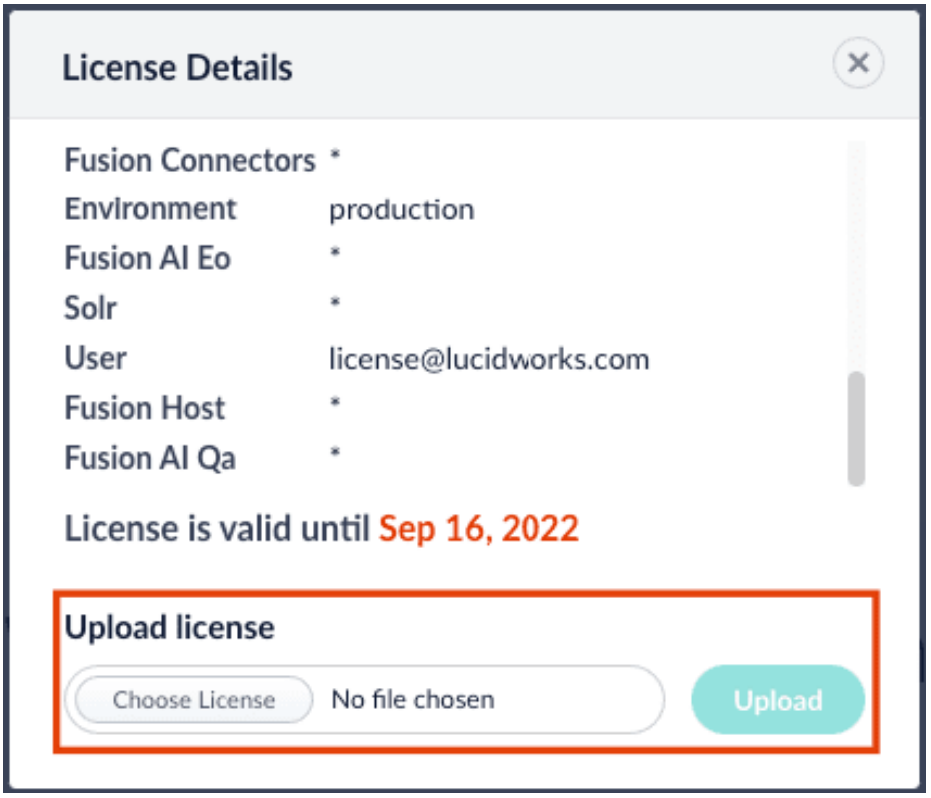
To verify your Experience Optimizer license, click the **Account Settings** icon.

and select **License Details**.

If you have a valid Experience Optimizer license, a * displays after **Fusion AI Eo**:



If you do not see a *, you must upload a valid license file to use Experience Optimizer:



Optimizing Zero Results

As a search manager, search queries that generate zero results can drive customers away from the site. Experience Optimizer allows you to ensure this does not happen. Instead of display zero results, you can choose to display certain docuents when a customer reaches a zero results page.

Learn how to use the *optimizing zero results* feature.

Rule Conflict Resolution

Sometimes, rules created in the *Rules Editor* and Experience Optimizer conflict with each other. For example, if you have 3 rules to boost, bury, and block a document, Experience Optimizer highlights these conflicting rules so you can address them.

To highlight conflicting rules, the **Rules Fired** section will display a red warning icon next to the dominant rule and a red flag on the conflicting rules.

By clicking the **Edit** button, you can view the details of the conflicting rules so the conflict can be resolved:

The details of the conflicting rules affecting specific products are found in the document details panel. Click the **Start Task** button

Hover over a document, and click the **Document Details** button.

Search Rewrites

Search rewriting is a way of modifying search queries so that they more accurately reflect the intentions of your customers. You can create a misspelling search rewrite, for example, to correct a spelling mistake that commonly occurs when searching for a particular document. The misspelled query will then produce the same results as a correctly spelled search.

There are four rewrite types:

- *Head/Tail* - improves poorly performing searches.
- *Misspelling detection* - corrects common spelling mistakes.
- *Phrase detection* - identifies products with matching phrases.
- *Synonym detection* - includes alternative words with the same meaning.

In Experience Optimizer, you can create search rewrites manually. You can also edit, test, review and publish the search rewrites that are generated automatically in Fusion from signals data.

Supported Browsers

Currently, Experience Optimizer is officially supported in Google Chrome *only*. Although Experience Optimizer is available in unsupported browsers, features and functionality may be limited.

Browser	Supported
Google Chrome	✅
Firefox	×
Safari	×
Opera	×
Edge	×

Data Analytics

- *Experiments* provide A/B testing and *metrics*.
- You can run and schedule *analytics jobs* to keep your analytics up to date.
- *App Insights* provides visualizations about your signals and experiments.
- *Fusion SQL* is a powerful engine for analyzing your Fusion data.

Querying Fusion SQL

Fusion SQL provides a set of search and analytic capabilities designed for data exploration and visualization.

Fusion SQL functions

- ***search***
- ***text analytics***
  - *Text Analysis and Analytics (`analyze`)*
  - *Significant Terms (`significant_terms`)*
- ***aggregation***
- ***time series***
- ***statistics***
  - *Co-occurrence Matrices (`co_matrix`)*
  - *Correlation Matrices (`corr_matrix`)*
  - *Cumulative Probability (`cumulative_probability`)*
  - *Gaussfit (`gaussfit`)*
  - *Geo Clustering (`geo_cluster`)*
  - *Histogram (`hist`)*
  - *2D K-Means Clustering (`kmeans`)*
  - *Knnfit (`knnfit`)*
  - *Regression and Residual Plot (`linearfit`)*
  - *Mann-Whitney (`mann_whitney`)*
  - *Percentiles (`percentiles`)*
  - *Polyfit (`polyfit`)*
  - *Probability (`probability`)*
  - `qqplot`
  - *Regression Diagnostics (`regress`)*
  - `ttest`
- ***miscellaneous***
  - `concat_text`
  - `timediff`
  - `weighted_sum`

Jupyter and SQL

Fusion 5 provides a Jupyter service that can query and visualize result sets and statistical functions from Fusion SQL. See these topics:

- *How to enable Jupyter*
- *How to use Jupyter with Fusion SQL*

2D K-Means Clustering (kmeans)

The `kmeans` function performs 2D k-means clustering. 2D k-means clustering can be used to visualize patterns within 2D scatter plots. The `kmeans` function takes three parameters:

1. The numeric field for the first dimension
2. The numeric field for the second dimension
3. K or number of clusters

Sample syntax

```
select kmeans(petal_length_d, petal_width_d, 5) as cluster, petal_length_d,
petal_width_d from iris limit 150
```

Result set

The result set contains a random sample of records that match the `WHERE` clause. If no `WHERE` clause is included, the random sample will be taken from the entire result set. The size of the random sample can be controlled by the `LIMIT` clause. The default sample size, if no limit is applied, is 25,000.

The `kmeans` function returns the cluster name of each row in the result set. The two fields used for clustering are also available in the result set.

Sample result set in Apache Zeppelin

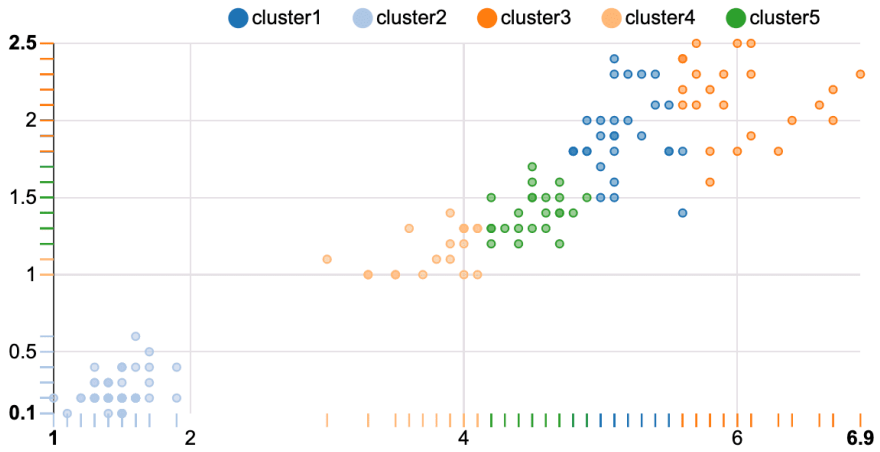
Visualization

Sample visualization of kmeans cluster with Apache Zeppelin scatter plot.

FINISHED

```
select kmeans(petal_length_d, petal_width_d, 5) as cluster,
       petal_length_d,
       petal_width_d
from iris
    limit 150
```

settings



Took 1 sec. Last updated by anonymous at July 30 2020, 10:51:15 AM. (outdated)

Co-occurrence Matrices (co_matrix)

The `co_matrix` function returns a matrix that shows the correlation of values within a categorical field based on their co-occurrence with another categorical field. For example, in a medical database this could be used to correlate diseases by co-occurring symptoms. In the example below, the `co_matrix` function is used to correlate complaint types across zip codes in the NYC 311 complaint database. This can be used to better understand how complaint types tend to go together. The `co_matrix` function has 4 parameters:

1. The categorical String field that will be correlated
2. The categorical String field that will be used for co-occurrence
3. Number of categorical variables to correlate
4. Number of categorical variables to calculate co-occurrence from

Sample syntax:

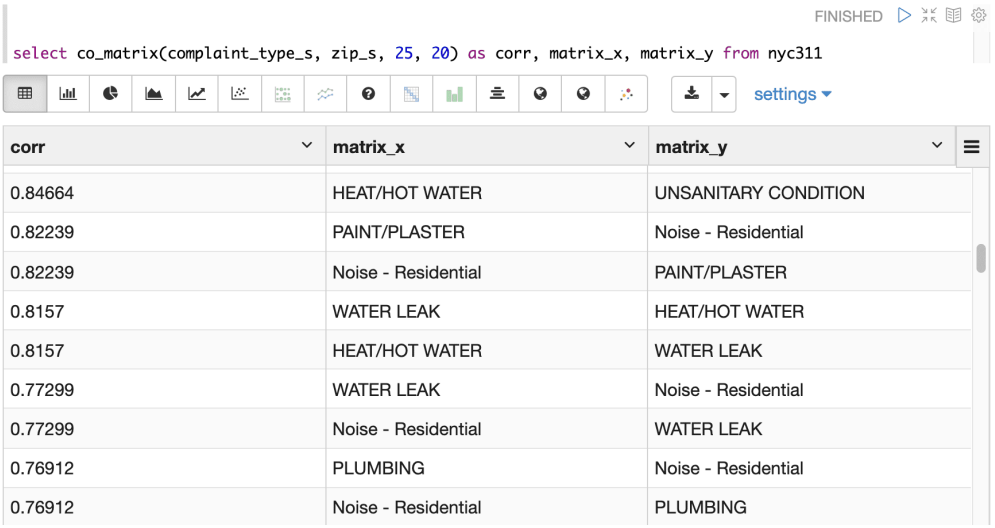
In the example below, the top 25 values in the `complaint_type_s` field are correlated across the top 20 values in the `zip_s` field in the NYC 311 complaint database.

```
select co_matrix(complaint_type_s, zip_s, 25, 20) as corr, matrix_x, matrix_y from
nyc311
```

Result set

The result set for the `co_matrix` function is a correlation matrix for the first categorical field parameter. The `co_matrix` function returns the correlation for each row. The `matrix_x` and `matrix_y` fields contain the combinations of the top N categorical values.

In the example below, Apache Zeppelin is used to display the correlation matrix result for the top 25 occurring values in the `complaint_type_s` field:



corr	matrix_x	matrix_y
0.84664	HEAT/HOT WATER	UNSANITARY CONDITION
0.82239	PAINT/PLASTER	Noise - Residential
0.82239	Noise - Residential	PAINT/PLASTER
0.8157	WATER LEAK	HEAT/HOT WATER
0.8157	HEAT/HOT WATER	WATER LEAK
0.77299	WATER LEAK	Noise - Residential
0.77299	Noise - Residential	WATER LEAK
0.76912	PLUMBING	Noise - Residential
0.76912	Noise - Residential	PLUMBING

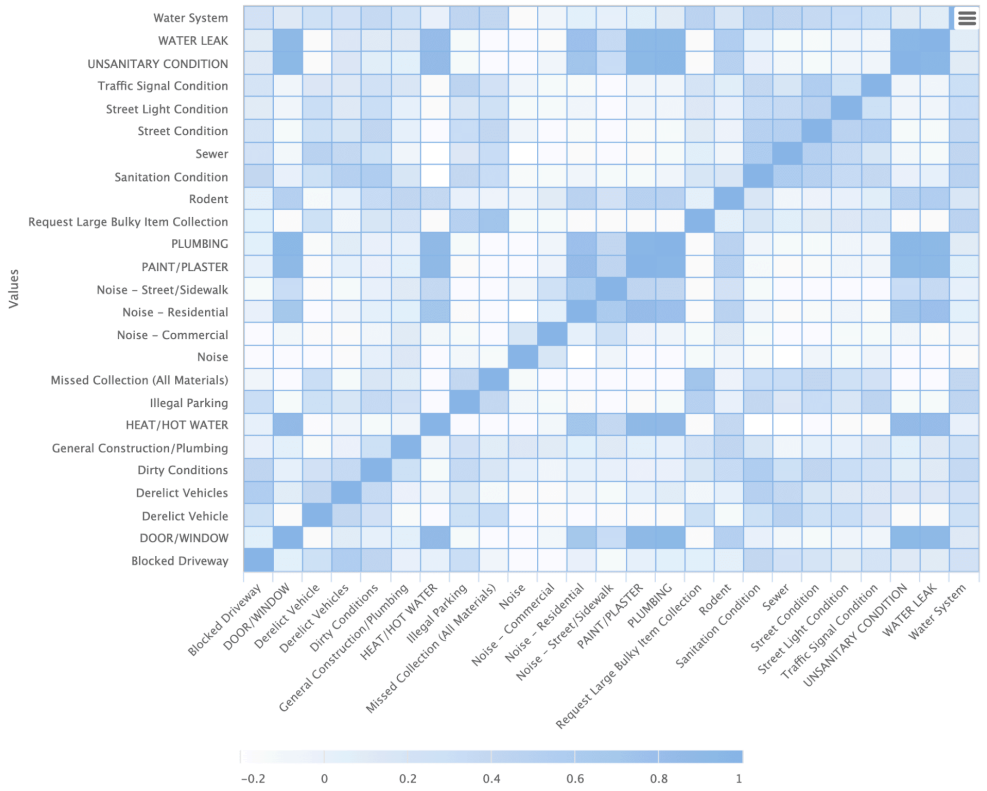
Visualization

The `co_matrix` function can be visualized in a heatmap by plotting `matrix_x` on the x-axis, `matrix_y` on the y-axis and the correlation value in the cells. The example below shows the `co_matrix` function visualized in an Apache Zeppelin heatmap:

```
select co_matrix(complaint_type_s, zip_s, 25, 20) as corr, matrix_x, matrix_y from nyc311
```

FINISHED

settings



Correlation Matrices (corr_matrix)

Correlation matrices can be computed using the `corr_matrix` function. The `corr_matrix` function takes two parameters:

1. A string, enclosed in single quotes, containing a comma-separated list of numeric fields for which to calculate the matrix
2. The sample size to compute the correlation matrix from

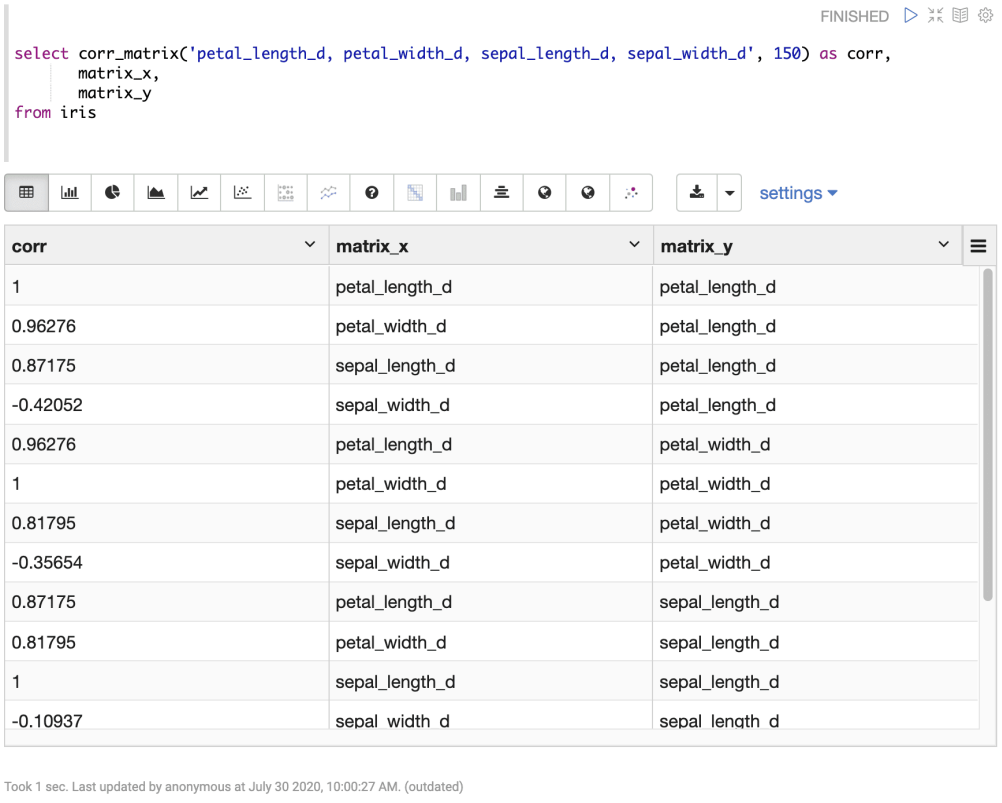
Sample syntax

```
select corr_matrix('petal_length_d, petal_width_d, sepal_length_d, sepal_width_d', 150) as corr, matrix_x, matrix_y from iris
```

Result set

The result set for the `corr_matrix` function contains one row for each two-field combination listed in the first parameter. The `corr_matrix` function returns the correlation for the two-field combination. There are two additional fields, `matrix_x` and `matrix_y` that contain the field combination for the row.

Sample result set in Apache Zeppelin



Visualization

The example below shows the `corr_matrix` result visualized in Apache Zeppelin with a heat map.

FINISHED

```
select corr_matrix('petal_length_d, petal_width_d, sepal_length_d, sepal_width_d', 150) as corr,
       matrix_x,
       matrix_y
from iris
```

Took 1 sec. Last updated by anonymous at July 30 2020, 10:00:27 AM. (outdated)

Cumulative Probability (cumulative_probability)

The `cumulative_probability` function calculates the probability that a value drawn from a numeric field will be less then or equal to a specific value. The `cumulative_probability` function takes three parameters:

1. The numeric field to compute the cumulative probability from
2. The specific numeric value to compute the cumulative probability for
3. The sample size

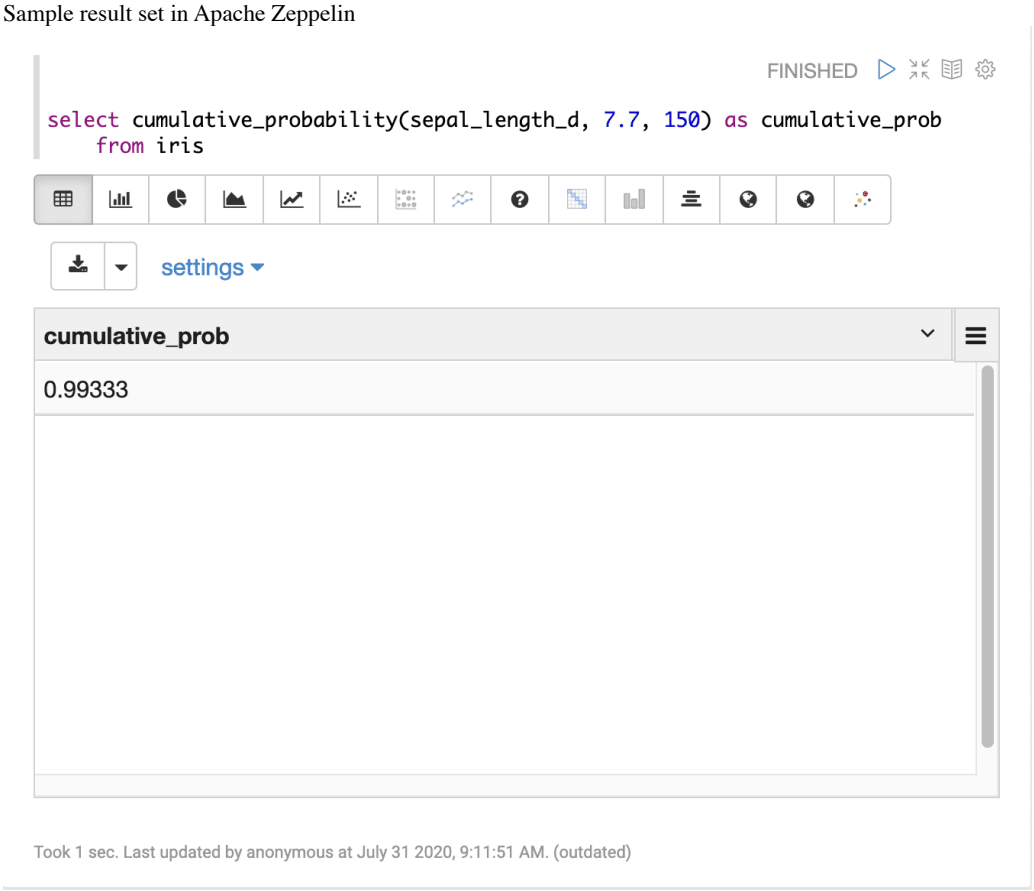
Sample syntax

```
select cumulative_probability(sepal_length_d, 7.7, 150) as cumulative_prob from
iris
```

Result set

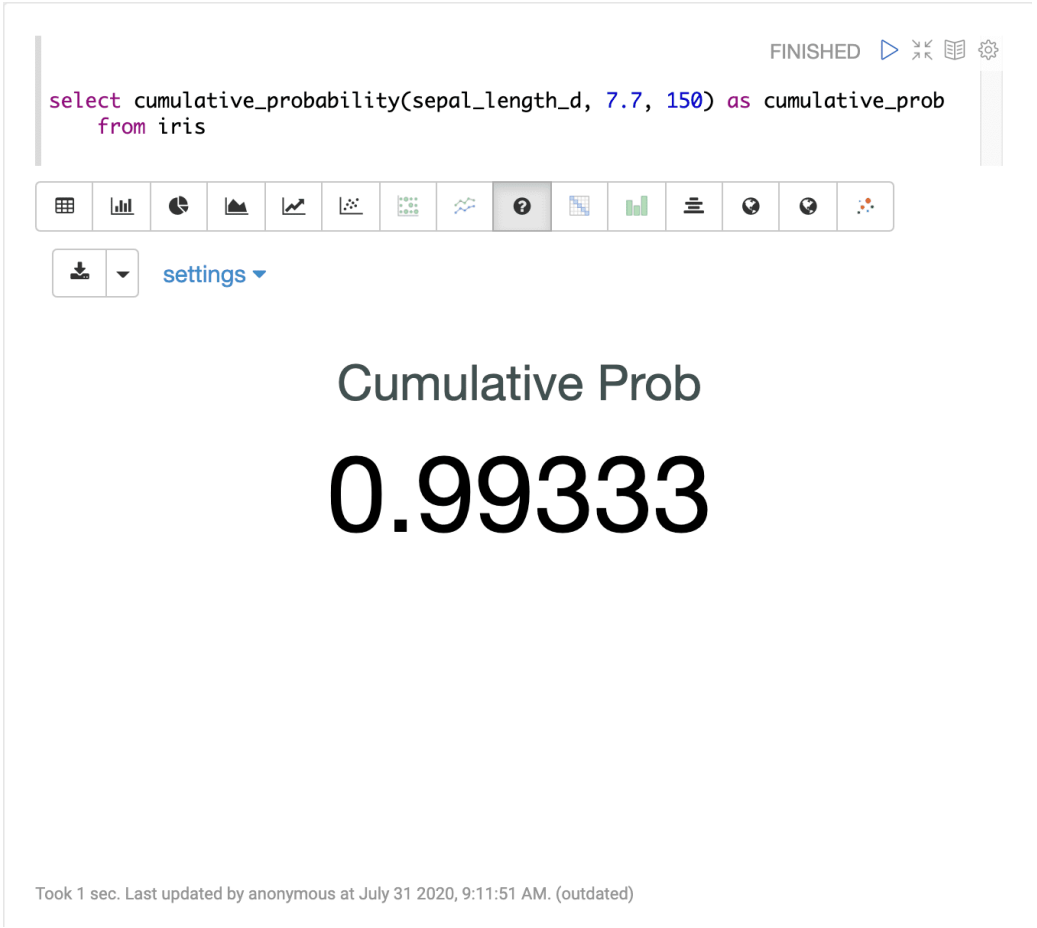
The result set for the `cumulative_probability` function contains a single record with the calculated cumulative probability. The `cumulative_probability` function returns the cumulative probability calculation.

Sample result set in Apache Zeppelin



Visualization

Sample visualization of the `cumulative_probability` function using the Apache Zeppelin number visualization:



Fusion SQL Aggregation

SQL aggregations are translated to Solr facet queries to take advantage of Solr's distributed aggregation capabilities. This allows for interactive data analysis over large data sets.

Single and multi-dimension aggregation using supported aggregation functions operate over the entire query result and are designed to return accurate results. The supported aggregation functions that are fully pushed down to the search engine are: `count(*)`, `count(distinct)`, `sum`, `avg`, `min`, `max`.

An example of a SQL aggregation that is translated to a Solr facet query is below:

```
select company_name, count(*) as cnt from orders group by company_name order by cnt desc
```

Having Clause

A `HAVING` clause can also be applied to single and multi-dimension aggregations.

Fusion SQL Miscellaneous Functions

These Fusion SQL functions provide miscellaneous features:

- *`click_pos`*
- *`concat_text`*
- *`timediff`*
- *`weighted_sum`*

Fusion SQL Search

Scoring

The WHERE and ORDER BY clauses can be used to search and sort results using the underlying search engine. The **score** (lower case) keyword can be used to sort by the relevance score of a full text query.

An example of a query that uses the **score** keyword is below:

```
select id, title, score from books where abstract = 'hello world' order by score
desc
```

Searching

Search predicates are specified in the WHERE clause. Search predicates on text fields will perform full text searches. Search predicates on string fields will perform exact matches unless the LIKE expression is used.

By default all **multi-term** predicates are sent to the search engine as phrase queries. In the example above 'hello world' is searched as a phrase query.

To stop the auto-phrasing of multi-term predicates wrap parenthesis around the terms. For example:

```
select id, title, score from books where abstract = '(hello world)' order by score
desc
```

In the example above the '(hello world)' search predicate will be sent to the search engine without phrasing and perform the query **hello OR world**

When parentheses are used the search expression is sent to Solr unchanged. This allows for richer search predicates such as proximity search.

The example below performs a proximity search:

```
select id, title, score from books where abstract = '("hello world"~4)' order by
score desc
```

Lucene/Solr wildcards can be sent to the search engine directly using this syntax:

```
select id, title from books where abstract = '(he?lo)'
```

The LIKE clause can be used to perform wildcard searches with either the Solr wildcard symbol * or the SQL wildcard symbol %.

When using the traditional SQL % wildcard only leading and trailing wildcards are supported. Use Lucene/Solr wildcards as described above for more complex wildcards.

The example below shows a LIKE query with a trailing % wildcard.

```
select id, title from books where abstract like 'worl%'
```

The following operators are supported for numeric and datetime predicates: <, >, >=, <=, =, !=.

Both the IN and BETWEEN clauses can be used to specify predicates.

Boolean predicates can be used and are translated to boolean search queries.

The example below specifies a boolean query:

```
select id from products where prod_desc = 'bike' and price < 125 order by price
asc
```

Sorting

Numeric, datetime and string fields can be sorted on using the ORDER BY clause. The sort is pushed down to the search engine for optimal performance. Multiple sorts can be specified using the standard SQL sytnax.

The example below sorts on a numeric field:

```
select id, prod_name, price_f from products where prod_desc = 'bike' order by
price_f desc
```

Fusion SQL Statistics

Sampling is often used in statistical analysis to gain an understanding of the **distribution**, **shape**, and **dispersion** of a variable or the relationship between variables.

Fusion SQL returns a random sample for all basic selects that **do not** contain an ORDER BY clause. The random sample is designed to return a uniform distribution of samples that match a query. The sample can be used to infer statistical information about the larger result set.

The example below returns a random sample of single field:

```
select filesize_d from logs where year_i = 2019
```

If no limit is specified the sample size will be 25000. To increase the sample size add a limit larger than 25000.

```
select filesize_d from logs where year_i = 2019 limit 50000
```

The ability to subset the data with a query and then sample from that subset is called **Stratified Random Sampling**. Stratified Random Sampling is an important statistical technique used to better understand sub-populations of a larger data set.

Descriptive statistics

Sub-queries can be used to return random samples for fast, often sub-second, statistical analysis. For example:

```
select count(*) as samplesize, mean(filesize_d) as mean, min(filesize_d) as min,
max(filesize_d) as max, approx_percentile(filesize_d, .50) as median,
variance(filesize_d) as variance, std(filesize_d) as standard_dev,
skewness(filesize_d) as skewness, kurtosis(filesize_d) as kurtosis, sum(filesize_d)
as sum from (select filesize_d from logs where year_i = 2019 limit 50000)
```

In the example above, the sub-query is returning a random sample of 50000 results which is operated on by the main statistical query. The statistical query returns aggregations which describe the distribution, shape, and dispersion of the sample set.

Correlation and covariance

Sub-queries can be used to provide random samples for correlation and covariance:

```
select corr(filesize_d, response_d) as correlation, covar_samp(filesize_d,
response_d) as covariance from (select filesize_d, response_d from logs limit
50000)
```

In the example above, the random sample returns two fields to the corr and covar_samp functions in the main query. Correlation and covariance are used to show the strength of the linear relationship between two variables.

Pushed-down statistical queries

A narrower set of statistical aggregations can be pushed down to the search engine and operate over entire result sets. These functions are: count(*), count(distinct), sum, min, max, avg and approx_percentile.

Below is an example of a fully pushed down statistical query:

```
select count(*) as cnt, avg(filesize_d) as avg, approx_percentile(filesize_d, .50)
as median from logs where year_i = 2018
```

Statistical queries that contain a mix of the queries above and non-pushdown such as skewness or kurtosis will be operate over a random sample that matches the query.

Below is an example of a statisical query that operates over a random sample:

```
select count(*) as cnt, skewness(filesize_d) as skewness from logs where year_i =
2018
```

Fusion SQL Text Analytics

These built-in functions are available for text analytics:

- *`significant_terms`*
  The `significant_terms` function finds anomaly terms in a text or string field that appear more frequently in a search result set than the entire index.
- *`analyze`*
  The `analyze` function analyzes a text or string field with a Lucene/Solr analyzer and returns the analyzed tokens. The use of Lucene/Solr analyzers at query time allows for real-time entity extraction by configuring OpenNLP, Stanford NLP, or other NLP analyzers.

Gaussfit (gaussfit)

The `gaussfit` function fits a gaussian curve to a histogram constructed from a random sample drawn from a numeric field. The gaussfit function can be used to visualize how well the values in a numeric field fit a normal distribution. The `gaussfit` function takes two parameters:

1. The numeric field from which to draw the histogram
2. The sample size

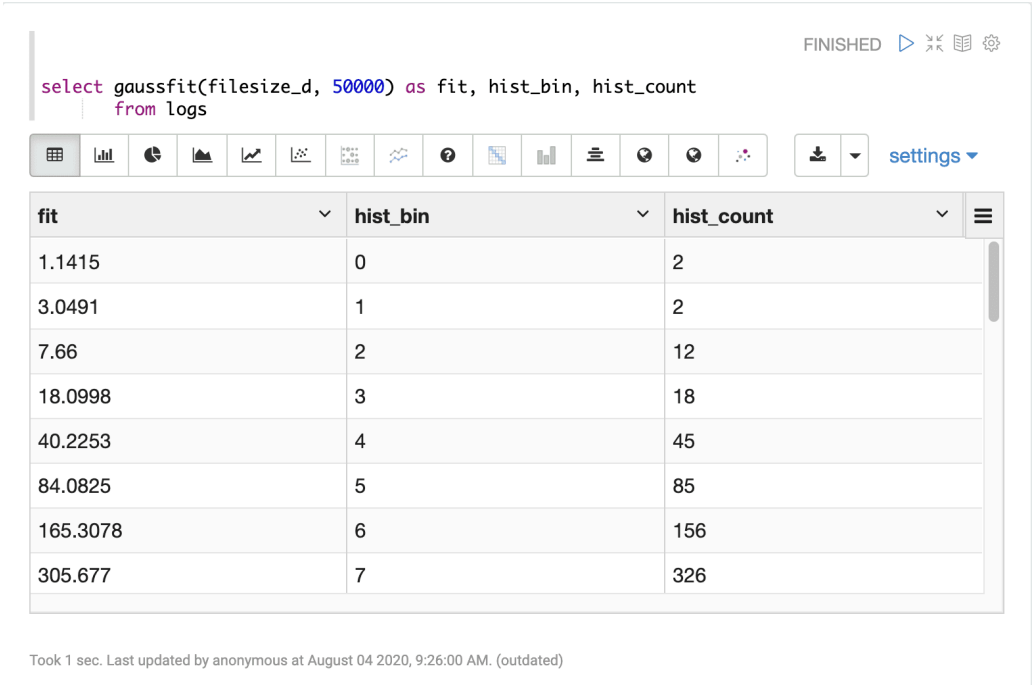
Sample syntax

```
select gaussfit(filesize_d, 50000) as fit, hist_bin, hist_count from logs
```

Result set

The gaussfit result set contains one record for each bin in the histogram drawn from the random sample. The `gaussfit` function returns the value of the fitted gaussian curve. The `hist_bin` and `hist_count` fields are also available in the gaussfit result set. The `hist_bin` field contains the histogram bin number and the `hist_count` field contains the count of samples in each bin.

Below is a sample result set in Apache Zeppelin:



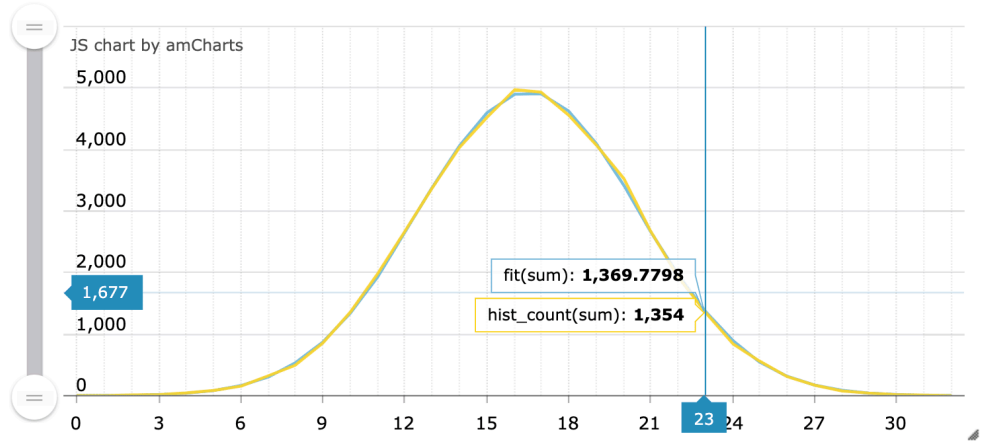
Visualization

The gaussfit result set can be visualized by plotting the `hist_bin` column on the x-axis and the `fit` and the `hist_count` columns on the y-axis. The visualization belows shows the gaussfit result visualized in an Apache Zeppelin line chart:

FINISHED

```
select gaussfit(filesize_d, 50000) as fit, hist_bin, hist_count
      from logs
```

settings



Took 1 sec. Last updated by anonymous at August 04 2020, 9:26:00 AM. (outdated)

Geo Clustering (geo_cluster)

The `geo_cluster` function performs geo-spatial clustering and noise reduction. The underlying algorithm used is DBSCAN clustering using haversine meters for the distance measure. The geo_cluster function takes four parameters:

1. The latitude field
2. The longitude field
3. The distance in meters to be considered a neighbor
4. The smallest number of points to be considered a cluster

Sample syntax

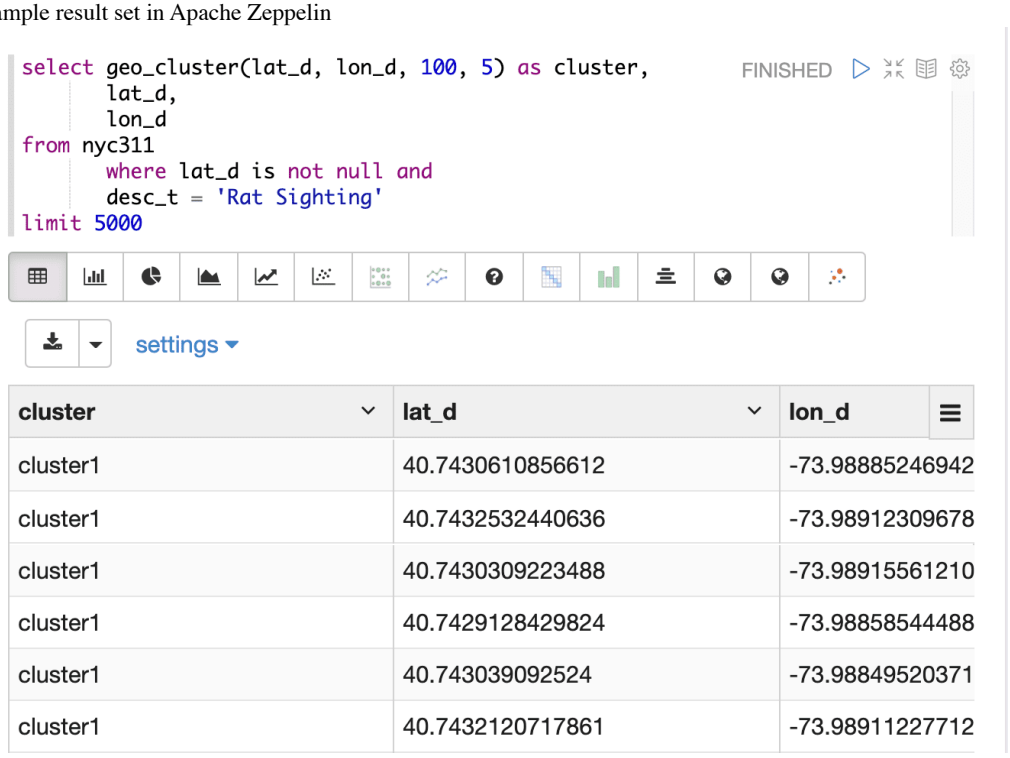
```
select geo_cluster(lat_d, lon_d, 100, 5) as cluster, lat_d, lon_d from nyc311 where lat_d is not null and desc_t = 'Rat Sighting' limit 5000
```

Result set

The `geo_cluster` result set contains a random sample of records that match the `WHERE` clause. If no `WHERE` clause is included the random sample will be taken from the entire result set. The size of the random sample can be controlled by the `LIMIT` clause. The default sample size, if no limit is applied, is 25,000. Points that match the `WHERE` clause but are not assigned to a cluster are not included in the result set. The noise-reduced result set makes it easy to quickly find hot spots or clusters in geo-spatial data. Due to the noise reduction the final result will likely be smaller than the limit.

The `geo_cluster` function returns the cluster name for each latitude/longitude point. The latitude/longitude point fields can also be selected for plotting.

Sample result set in Apache Zeppelin

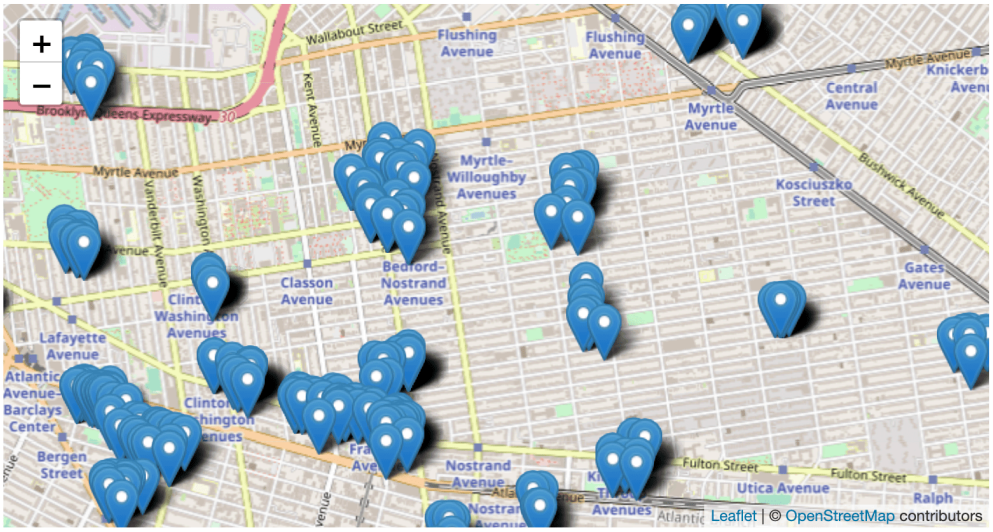


Visualization

The `geo_cluster` output can be visualized on a map or scatter plot. The example below shows the `geo_cluster` output visualized with an Apache Zeppelin map visualization.

```
select geo_cluster(lat_d, lon_d, 100, 5) as cluster,
       lat_d,
       lon_d
from nyc311
       where lat_d is not null and
       desc_t = 'Rat Sighting'
limit 5000
```

FINISHED



Took 2 sec. Last updated by anonymous at August 03 2020, 1:25:55 PM. (outdated)

Histogram (hist)

Numeric histograms can be created using the `hist` function. The `hist` function takes three parameters:

1. The numeric field to create the histogram from.
2. The number of bins in the histogram.
3. The sample size to create the histogram from.

Sample syntax

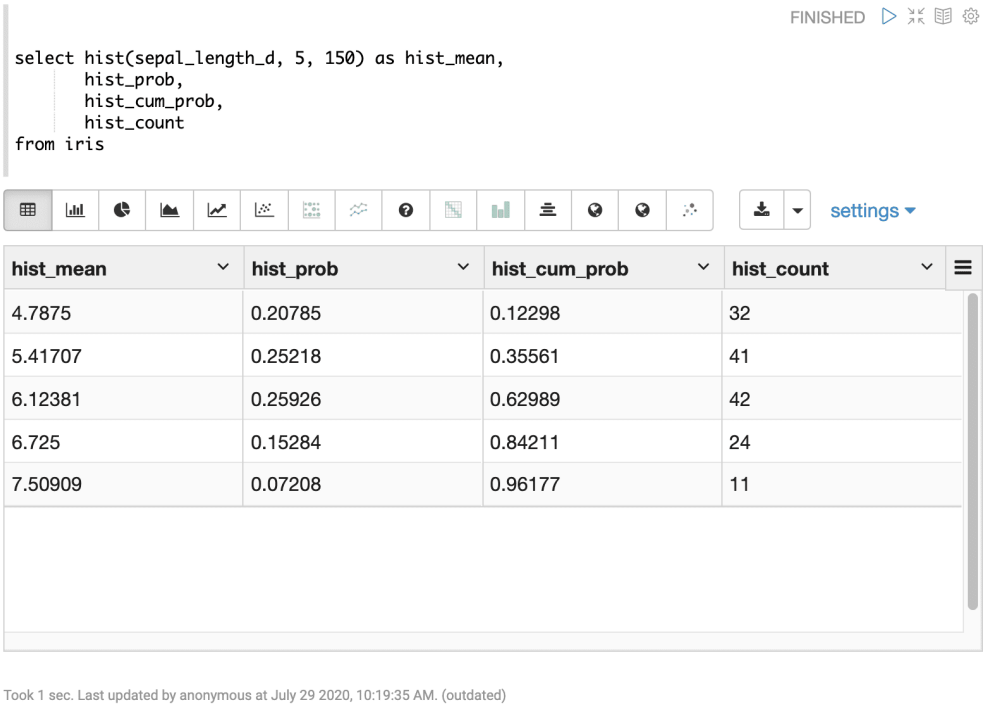
```
select hist(sepal_length_d, 5, 150) as hist_mean, hist_prob, hist_cum_prob,
hist_count from iris where species_s = 'setosa'
```

Result set

The result set from the histogram will contain one row for each histogram bin. The random sample for the histogram will be drawn from the results that match the WHERE clause in the SQL query. If no WHERE clause is provided, the samples will be drawn from the full data set. The `hist` function returns the mean of each bin. There are three additional fields that can be selected when the `hist` function is used:

- `hist_count` : the number of results within each bin.
- `hist_prob`: the probability of the bin, or the percentage of records within each bin.
- `hist_cum_prob`: the cumulative probability of each bin.

Sample result set shown in Apache Zeppelin Table:



Took 1 sec. Last updated by anonymous at July 29 2020, 10:19:35 AM. (outdated)

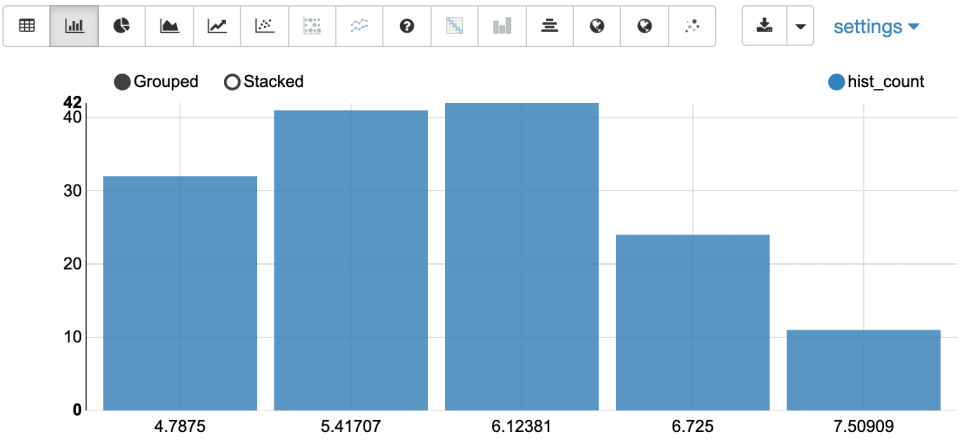
Visualization

Histograms can be visualized by plotting the bin means on the x-axis and either the `hist_count`, `hist_prob` or `hist_cum_prob` on the y-axis.

The example below shows a bar chart of the bin means and `hist_count` in Apache Zeppelin:

FINISHED

```
select hist(sepal_length_d, 5, 150) as hist_mean,
       hist_prob,
       hist_cum_prob,
       hist_count
from iris
```



Took 1 sec. Last updated by anonymous at July 29 2020, 10:19:35 AM. (outdated)

Knnfit (knnfit)

The `knnfit` function uses nearest neighbor regression to predict a curve through a bi-variate scatter plot. Nearest neighbor regression predicts based on the average of the k nearest neighbors in the sample set. This produces a curve that becomes smoother as k increases. The `knnfit` function takes three parameters:

1. The numeric field containing the independent (x) variable
2. The numeric field containing the dependent (y) variable
3. The number for k, the number of nearest neighbors from which to compute the average

Sample syntax

```
select knnfit(petal_length_d, petal_width_d, 4) as prediction, residual,
petal_length_d, petal_width_d from iris limit 150
```

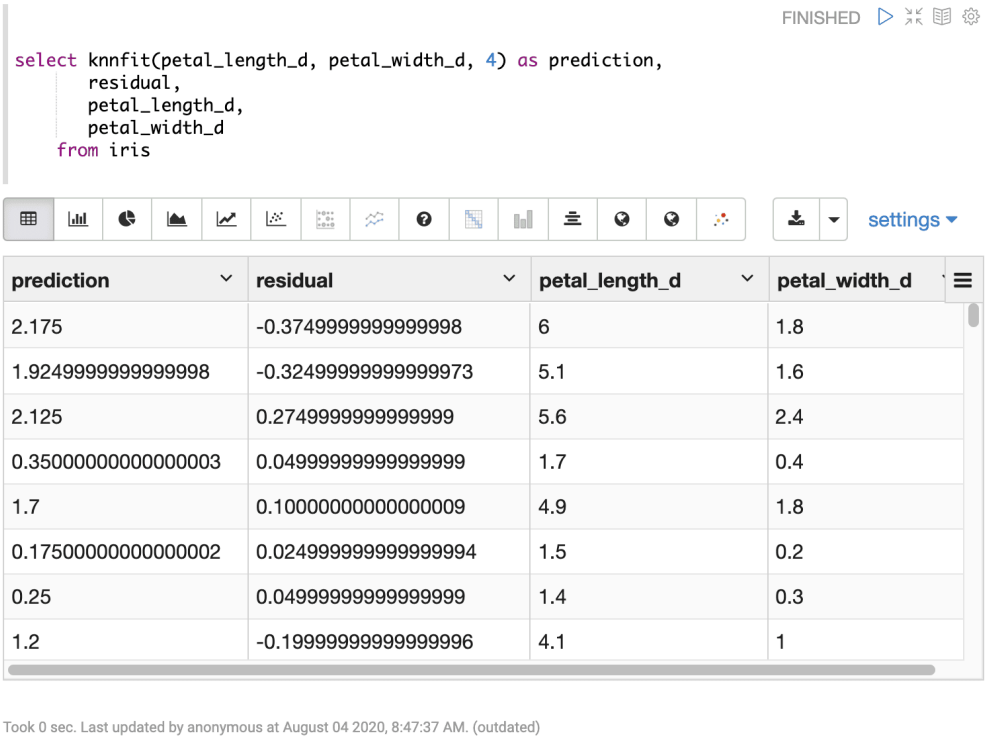
Result set

The result set contains a random sample of records that match the `WHERE` clause. If no `WHERE` clause is included, the random sample will be taken from the entire result set. The size of the result set can be controlled by the `LIMIT` clause. The default size, if no limit is applied, is 25,000.

The `knnfit` function returns the predicted value for each record. There are three additional fields that can be selected when the `knnfit` function is used:

- `residual`: the residual value for each sample
  The residual value is the sample's dependent (y) minus the predicted value. The residual represents the error of the regression prediction for each sample.
- the independent variable for each sample
- the dependent variable for each sample

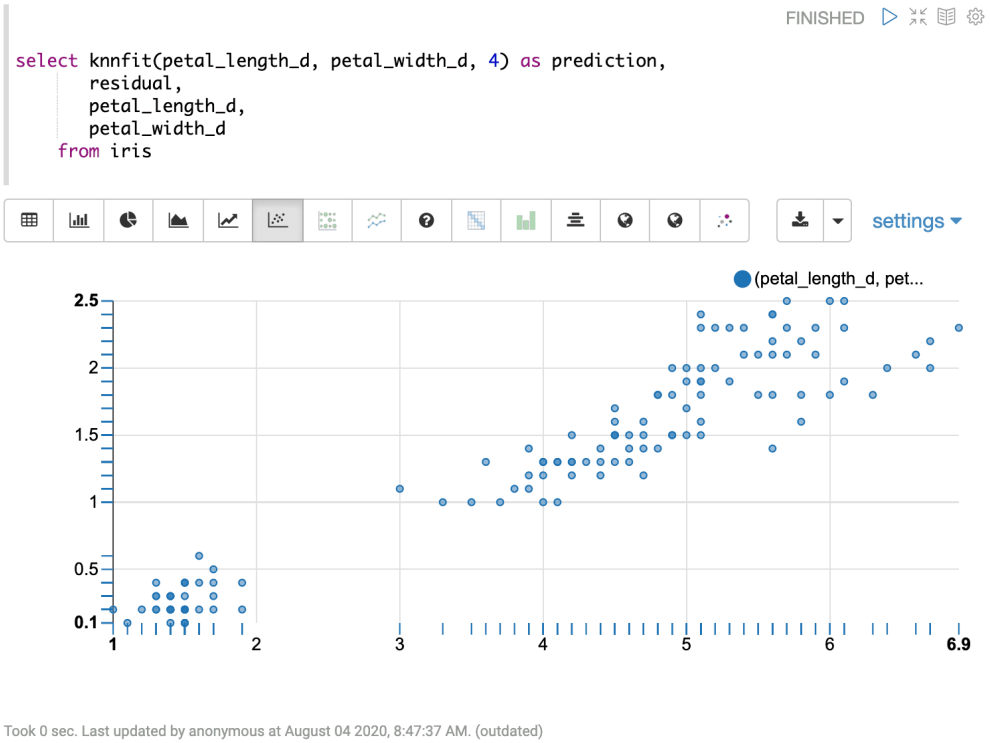
Sample result set in Apache Zeppelin



Visualization

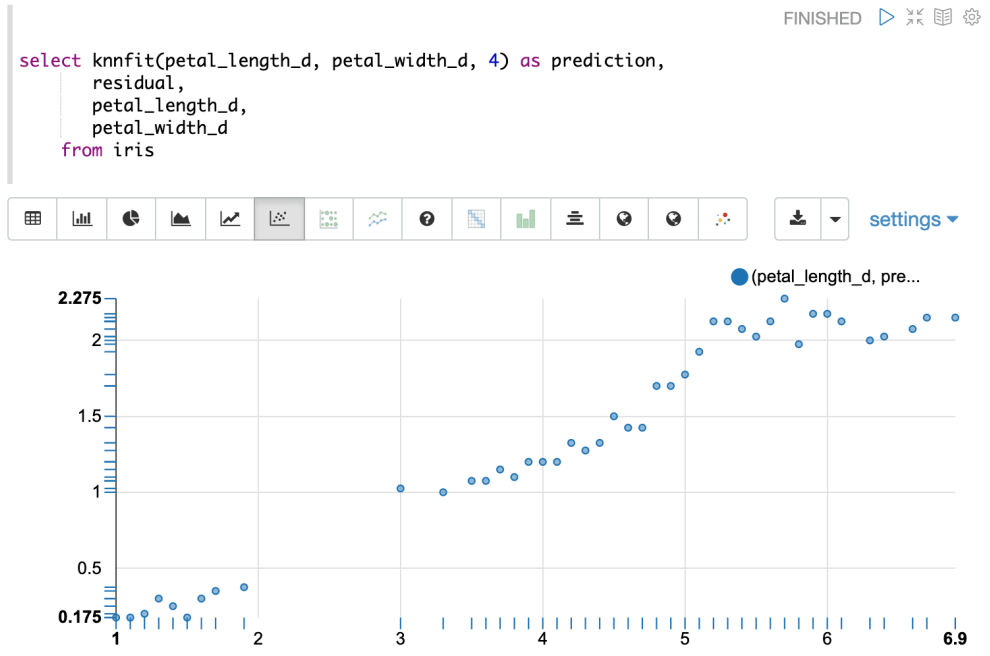
There are a number of visualizations that can flow from the regression result set.

The first visualization shown is a scatter plot with petal_length_d on the x-axis and petal_width_d on the y-axis. This can be used to visualize the relationship between the two variables in the regression analysis.



Took 0 sec. Last updated by anonymous at August 04 2020, 8:47:37 AM. (outdated)

The second visualization shows the petal_length_d variable on the x-axis and the prediction for petal_width_d on y-axis.

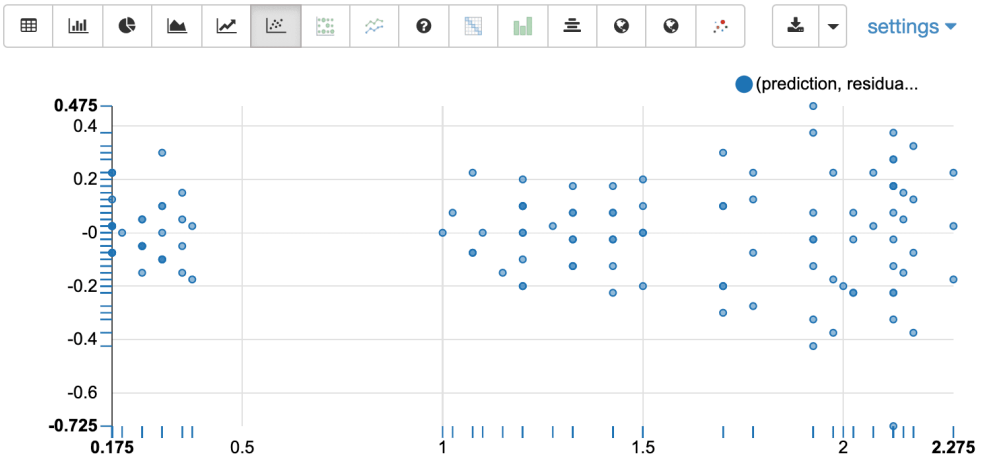


Took 0 sec. Last updated by anonymous at August 04 2020, 8:47:37 AM. (outdated)

The last visualization plots the predictions on the x-axis and the residual on the y-axis. This residual plot can be used to visualize the error of the regression model across the full range of predictions.

FINISHED

```
select knnfit(petal_length_d, petal_width_d, 4) as prediction,
       residual,
       petal_length_d,
       petal_width_d
    from iris
```



Took 0 sec. Last updated by anonymous at August 04 2020, 8:47:37 AM. (outdated)

Mann-Whitney (mann_whitney)

The `mann_whitney` function performs a Mann-Whitney U-test on two random samples drawn with different queries. The `mann_whitney` function takes four parameters:

1. The numeric field to perform the Mann-Whitney U-test on
2. The Lucene/Solr query for drawing the first sample
3. The Lucene/Solr query for drawing the second sample
4. The sample size for both samples

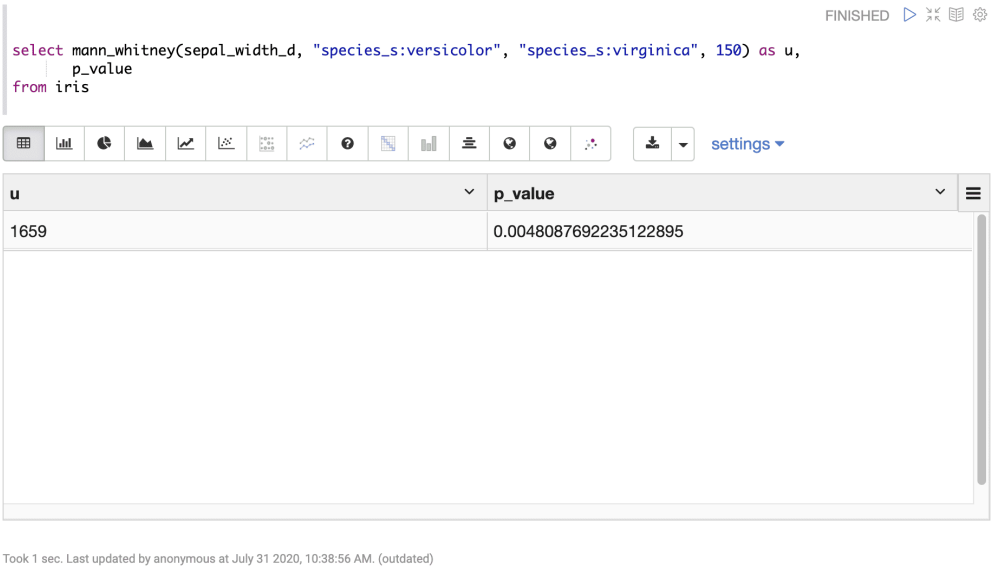
Sample syntax

```
select mann_whitney(sepal_width_d, "species_s:versicolor", "species_s:virginica",
150) as u, p_value from iris
```

Result set

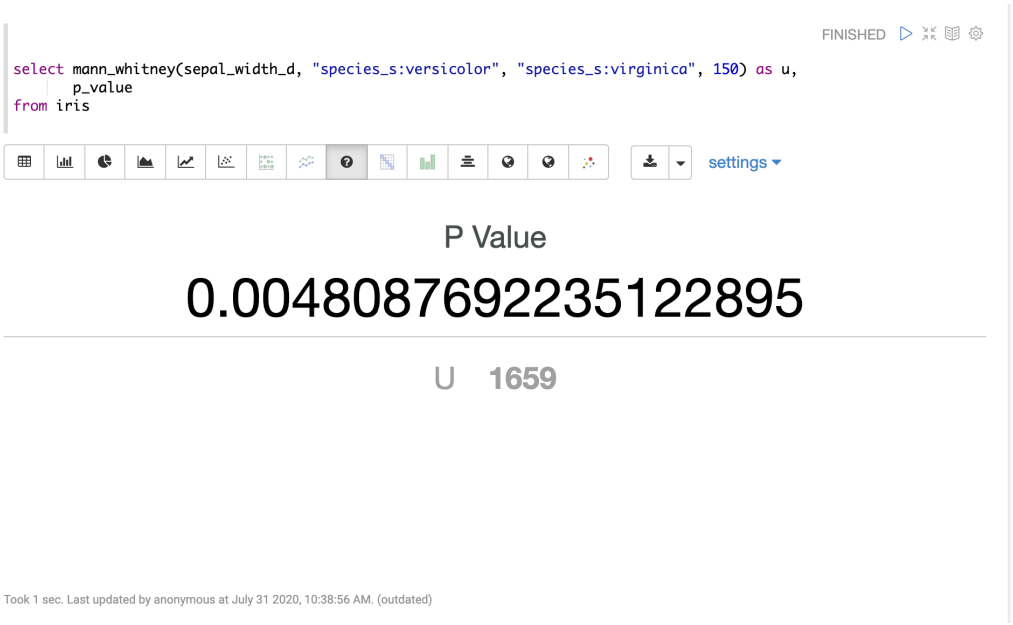
The result set for the `mann_whitney` function contains one row with the results of the `mann_whitney` test. The `mann_whitney` function returns the U-statistic. The `p_value` field can be selected to return the p-value for the Mann-Whitney test.

Sample result set in Apache Zeppelin



Visualization

Sample visualization of the `mann_whitney` function using the Apache Zeppelin number visualization:


FINISHED
select mann_whitney(sepal_width_d, "species_s:versicolor", "species_s:virginica", 150) as u,
p_value
from iris
settings
P Value
0.004808769223512289 5
U 1659
Took 1 sec. Last updated by anonymous at July 31 2020, 10:38:56 AM. (outdated)

Percentiles (percentiles)

The `percentiles` function can be used to visualize the estimated percentiles (1-99) for a numeric field. The `percentiles` function takes two parameters:

1. The numeric field to compute the percentiles for
2. The sample size

Sample Syntax

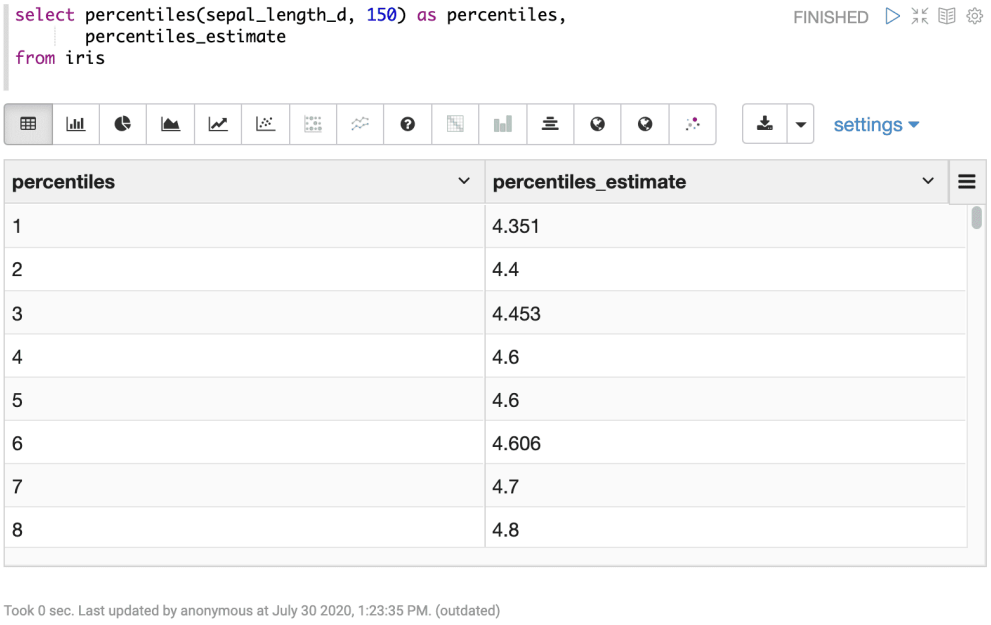
```
select percentiles(sepal_length_d, 150) as percentiles, percentiles_estimate from iris
```

Result set

The result set for the `percentiles` function contains one row for each percentile 1-99. The `percentiles` function returns the percentile (1-99). The `percentiles_estimate` field contains the estimated percentile value at each percentile.

The percentiles are calculated for a random sample that matches the `WHERE` clause. If there is no `WHERE` clause, the samples are taken from the entire data set.

Sample result set in Apache Zeppelin



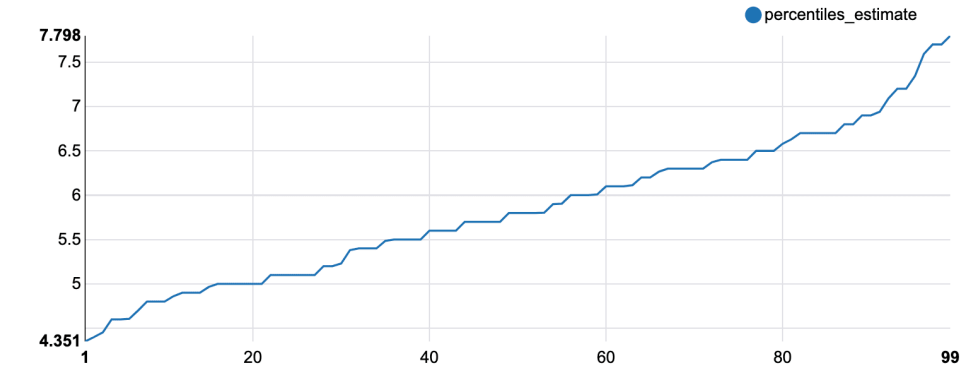
Visualization

The percentiles can be visualized by plotting the percentile (1-99) on the x-axis and the percentile estimate on the y-axis.

Sample visualization of percentiles in Apache Zeppelin:

```
select percentiles(sepal_length_d, 150) as percentiles,
       percentiles_estimate
from iris
```

FINISHED



Took 0 sec. Last updated by anonymous at July 30 2020, 1:23:35 PM. (outdated)

Polyfit (polyfit)

The `polyfit` function uses polynomial regression to predict a smooth, non-linear curve through a bi-variate scatter plot. The `polyfit` function takes three parameters:

1. The numeric field containing the independent (x) variable
2. The numeric field containing the dependent (y) variable
3. The integer degree of the polynomial
   The degree determines the number of curves to the fitted curve. One degree polynomial performs linear regression. Typically the degree is an integer from 1 to 5.

Sample syntax

```
select polyfit(petal_length_d, petal_width_d, 3) as prediction, residual,
petal_length_d, petal_width_d from iris limit 150
```

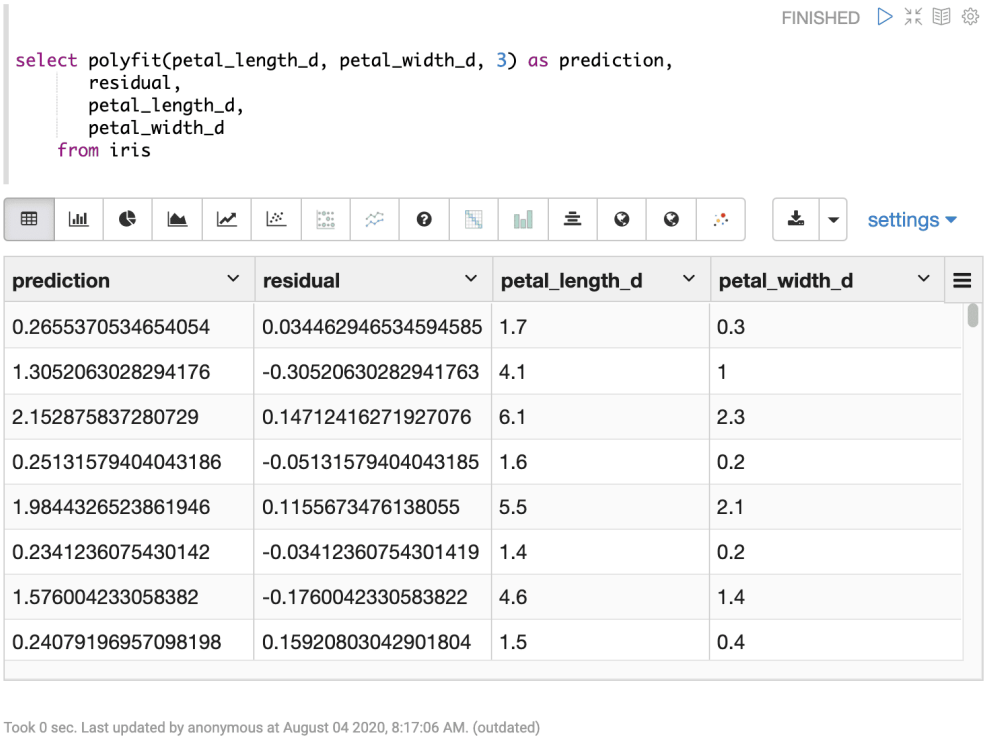
Result set

The result set contains a random sample of records that match the `WHERE` clause. If no `WHERE` clause is included the random sample will be taken from the entire result set. The size of the result set can be controlled by the `LIMIT` clause. The default size, if no limit is applied, is 25000.

The `polyfit` function returns the predicted value for each record. There are three additional fields that can be selected when the `polyfit` function is used:

- `residual`: the residual value for each sample
  The residual value is the samples dependent (y) value minus the predicted value. The residual represents the error of the regression prediction for each sample.
- the independent variable for each sample
- the dependent variable for each sample

Sample result set in Apache Zeppelin



prediction	residual	petal_length_d	petal_width_d
0.2655370534654054	0.034462946534594585	1.7	0.3
1.3052063028294176	-0.30520630282941763	4.1	1
2.152875837280729	0.14712416271927076	6.1	2.3
0.25131579404043186	-0.05131579404043185	1.6	0.2
1.9844326523861946	0.1155673476138055	5.5	2.1
0.2341236075430142	-0.03412360754301419	1.4	0.2
1.576004233058382	-0.1760042330583822	4.6	1.4
0.24079196957098198	0.15920803042901804	1.5	0.4

Took 0 sec. Last updated by anonymous at August 04 2020, 8:17:06 AM. (outdated)

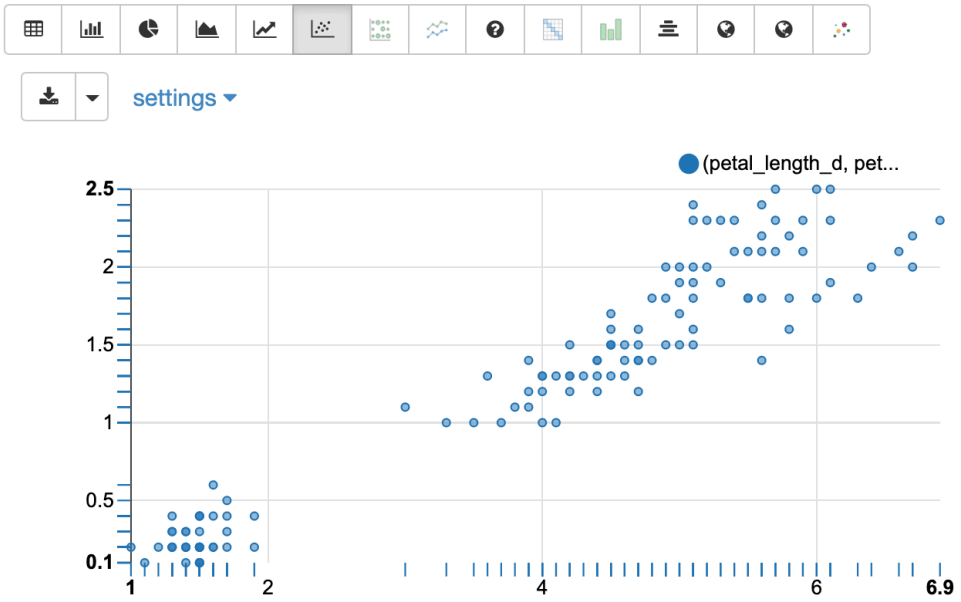
Visualization

There are a number of visualizations that can flow from the regression result set.

The first visualization shown is a scatter plot with `petal_length_d` on the x-axis and `petal_width_d` on the y-axis. This can be used to visualize the relationship between the two variables in the regression analysis.

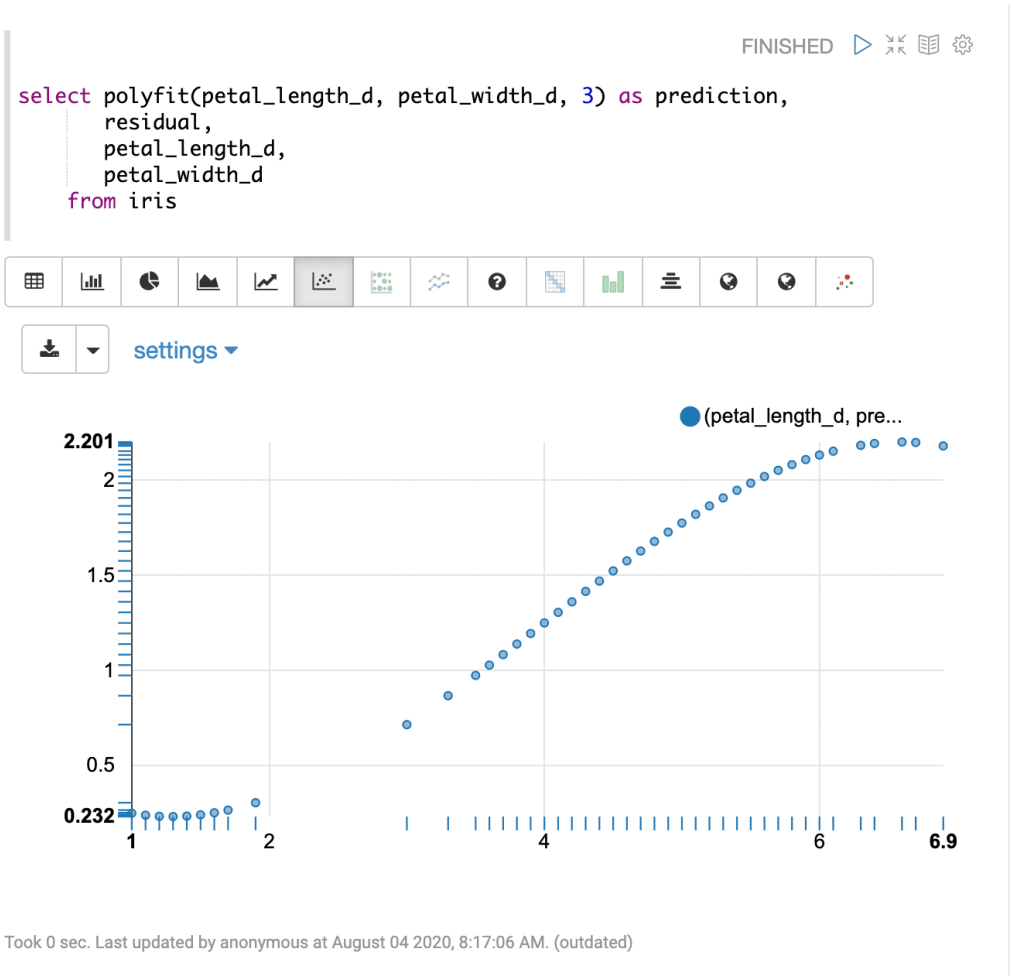
```
select polyfit(petal_length_d, petal_width_d, 3) as prediction,
       residual,
       petal_length_d,
       petal_width_d
   from iris
```

FINISHED



Took 0 sec. Last updated by anonymous at August 04 2020, 8:17:06 AM. (outdated)

The second visualization shows the `petal_length_d` variable on the x-axis and the prediction for `petal_width_d` on y-axis.

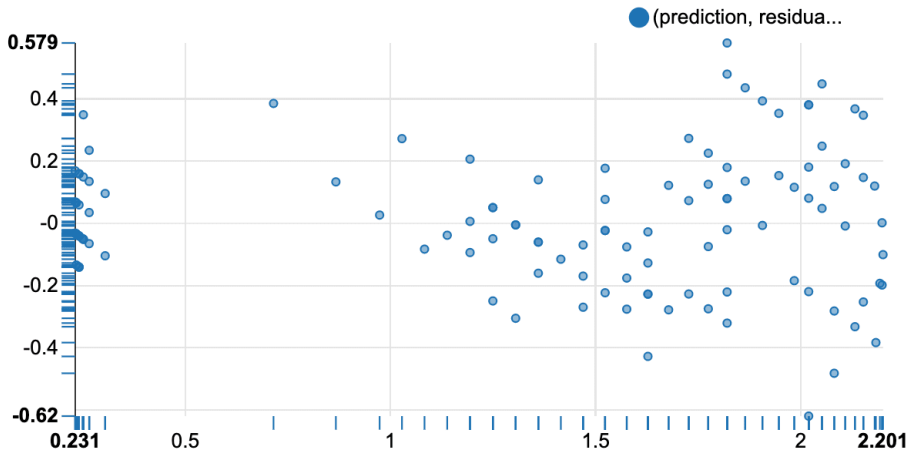


The last visualization plots the predictions on the x-axis and the residual on the y-axis. This residual plot can be used to visualize the error of the regression model across the full range of predictions.

FINISHED

```
select polyfit(petal_length_d, petal_width_d, 3) as prediction,
       residual,
       petal_length_d,
       petal_width_d
    from iris
```

settings



Took 0 sec. Last updated by anonymous at August 04 2020, 8:17:06 AM. (outdated)

Probability (probability)

The `probability` function calculates the probability that a value drawn from a numeric field will fall within specific range. The `probability` function takes four parameters:

1. The numeric field to compute the probability from
2. The numeric start range
3. The numeric end range
4. The sample size

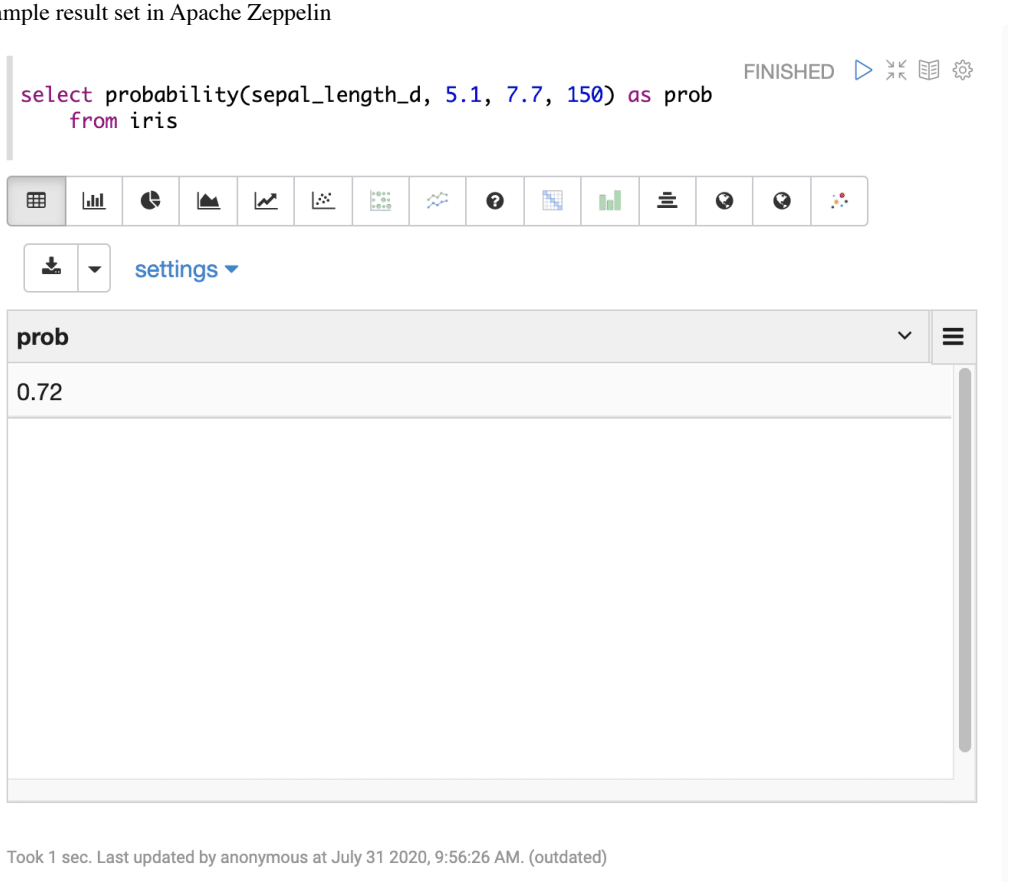
Sample syntax

```
select probability(sepal_length_d, 5.1, 7.7, 150) as prob from iris
```

Result set

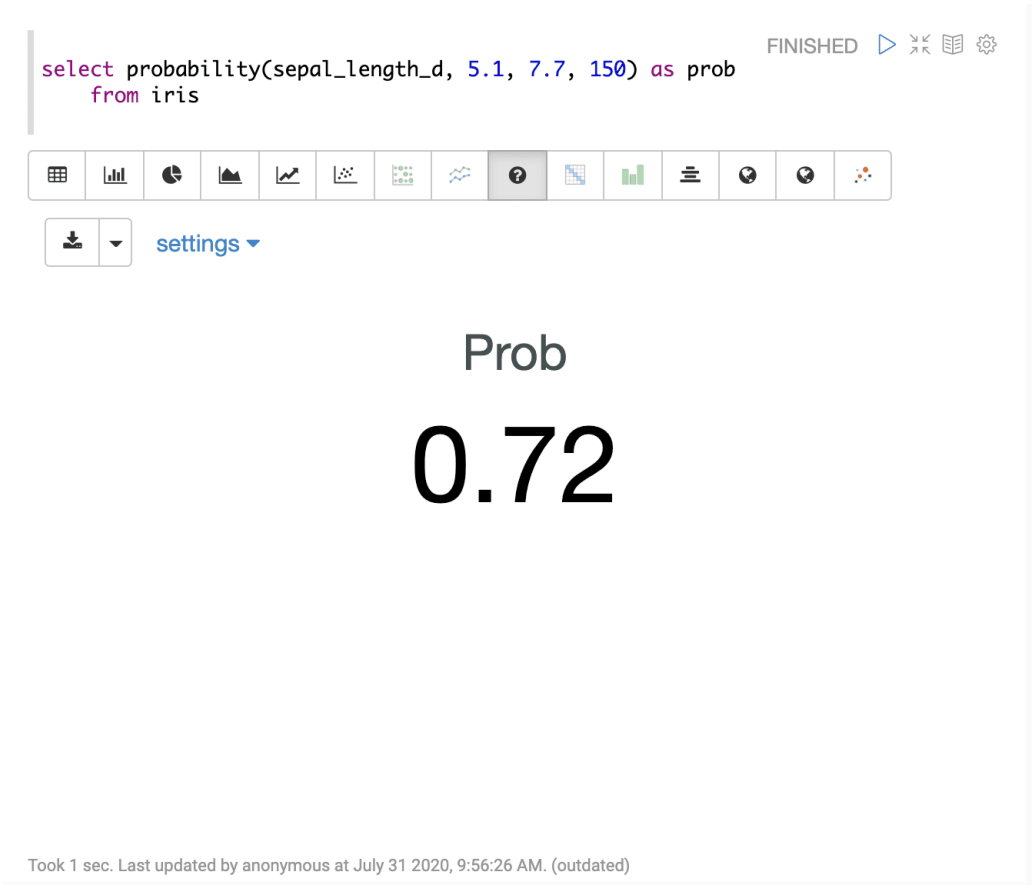
The result set for the `probability` function contains a single record with the calculated probability. The `probability` function returns the probability calculation.

Sample result set in Apache Zeppelin



Visualization

Sample visualization of the probability function using the Apache Zeppelin number visualization:


FINISHED
select probability(sepal_length_d, 5.1, 7.7, 150) as prob
from iris
settings
Prob
0.72
Took 1 sec. Last updated by anonymous at July 31 2020, 9:56:26 AM. (outdated)

QQPlot

The `qqplot` function calculates percentiles (1-99) for two sample sets, based on queries, so they can be compared visually. The `qqplot` function takes four parameters:

1. The numeric field for which to calculate the percentiles
2. The Lucene/Solr query for the sample set A
3. The Lucene/Solr query for the sample set B
4. The sample size for both sample sets

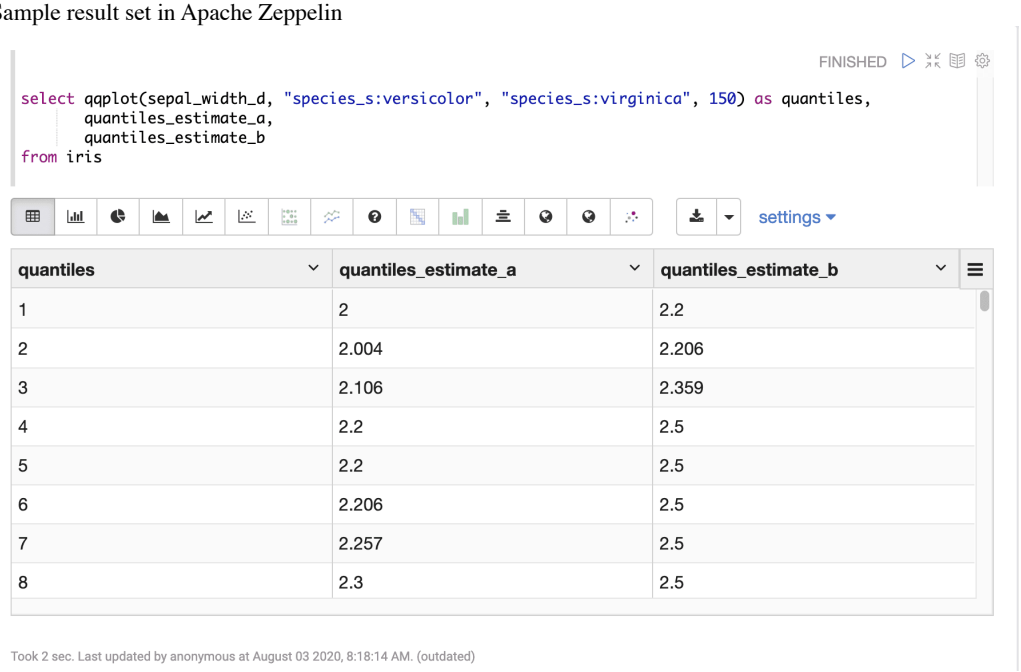
Sample syntax

```
select qqplot(sepal_width_d, "species_s:versicolor", "species_s:virginica", 150) as quantiles, quantiles_estimate_a, quantiles_estimate_b from iris
```

Result set

The result set for the `qqplot` function contains one row for each percentile 1-99. The `qqplot` function returns the percentile (1-99). The `quantiles_estimate_a` and `quantiles_estimate_b` fields contain the estimated percentile value, for sample sets A and B, at each percentile.

Sample result set in Apache Zeppelin



Visualization

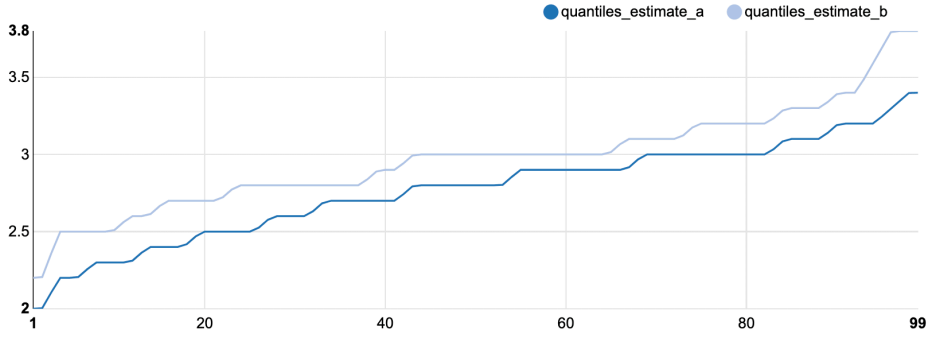
The qqplot function can be visualized by plotting the percentile (1-99) on the x-axis and the quantiles_estimate_a and quantiles_estimate_b columns on the y-axis.

Sample visualization of percentiles in Apache Zeppelin

FINISHED

```
select qqplot(sepal_width_d, "species_s:versicolor", "species_s:virginica", 150) as quantiles,
       quantiles_estimate_a,
       quantiles_estimate_b
from iris
```

settings



Took 2 sec. Last updated by anonymous at August 03 2020, 8:18:14 AM. (outdated)

Regression Diagnostics (regress)

The `regress` function computes diagnostics for bi-variate linear regression. The `regress` function takes three parameters:

1. The numeric field of the independent variable (x).
2. The numeric field of the dependent variable (y).
3. The sample size of the regression.

Sample syntax

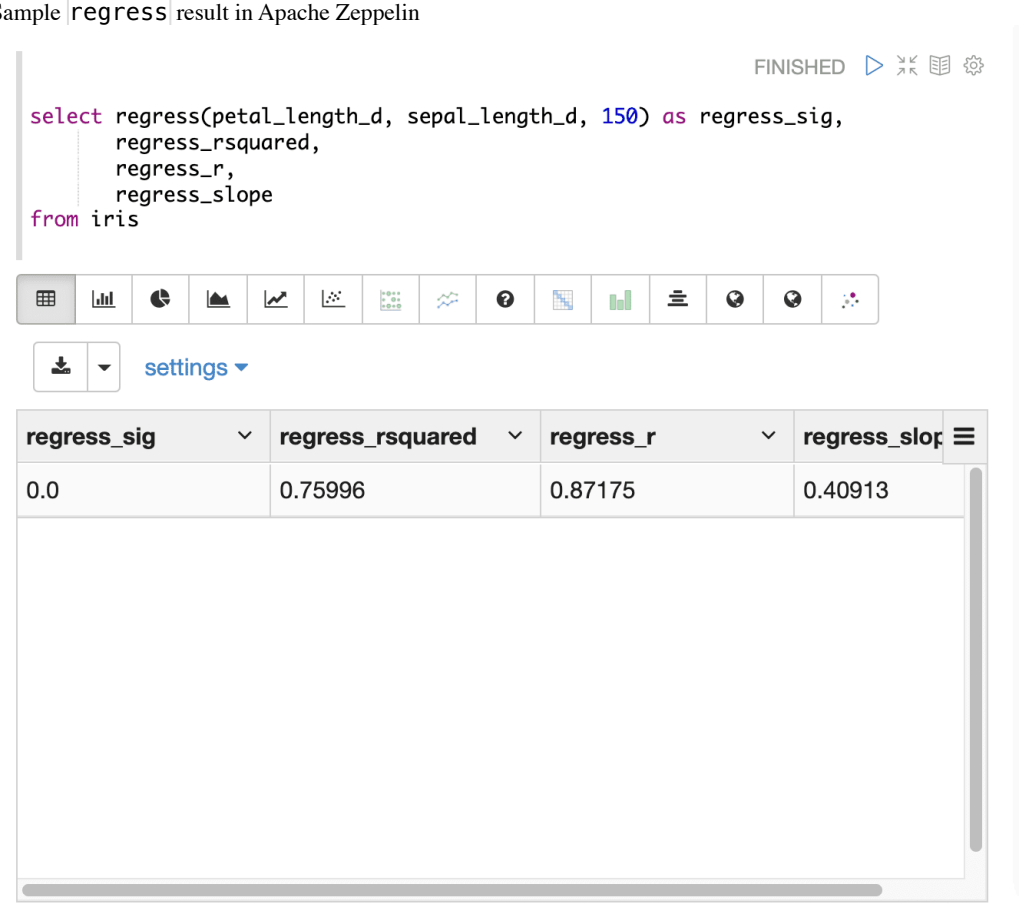
```
select regress(petal_length_d, sepal_length_d, 150) as regress_sig,
regress_rsquared, regress_r, regress_slope from iris
```

Result set

The result set for the `regress` function has one record that contains the selected regression diagnostics. The `regress` function returns the statistical significance of the regression analysis. The following regression diagnostics can be selected as well:

- `regress_slope` (slope)
- `regress_intercept` (y-intercept)
- `regress_rsquared` (R Squared)
- `regress_r` (correlation coefficient)
- `regress_mse` (mean square error)
- `regess_sse` (sum square error)
- `regress_ssr` (sum square due to regression)
- `regress_ssto` (total sum of squares)

Sample `regress` result in Apache Zeppelin



Took 0 sec. Last updated by anonymous at July 30 2020, 2:13:16 PM. (outdated)

Visualization

Sample visualization of the `regress` function using Apache Zeppelin's Number visualization.

Regression and Residual Plot (linearfit)

The `linearfit` function uses linear regression to predict a straight line through a bi-variate scatter plot. The `linearfit` function takes two parameters:

1. The numeric field containing the independent (x) variable.
2. The numeric field containing the dependent (y) variable.

Sample syntax

```
select linearfit(petal_length_d, petal_width_d) as prediction, residual,
petal_length_d, petal_width_d from iris limit 150
```

Result set

The result set contains a random sample of records that match the `WHERE` clause. If no `WHERE` clause is included, the random sample will be taken from the entire result set. The size of the random sample can be controlled by the `LIMIT` clause. The default sample size, if no limit is applied, is 25,000.

The `linearfit` function returns the predicted value for each record. There are three additional fields that can be selected when the `linearfit` function is used:

- `residual`: The residual value for each sample. The residual value is the sample's dependent (y) value minus the predicted value. The residual represents the error of the regression prediction for each sample.
- The independent variable for each sample.
- The dependent variable for each sample.

Sample result set shown in Apache Zeppelin

```
select linearfit(petal_length_d, petal_width_d) as prediction,
       residual,
       petal_length_d,
       petal_width_d
   from iris
```

FINISHED

settings

prediction	residual	petal_length_d	petal_width_d
1.8405073558958986	0.4594926441041012	5.3	2.3
1.3408043971534171	-0.34080439715341715	4.1	1
1.7572235294388183	0.04277647056118172	5.1	1.8
1.7572235294388183	0.1427764705611816	5.1	1.9
1.2991624839248772	8.37516075122835E-4	4	1.3
0.34139847966845466	0.058601520331545365	1.7	0.4
2.1320007484956793	-0.3320007484956793	6	1.8
1.7155816162102782	-0.015581616210278204	5	1.7

Took 0 sec. Last updated by anonymous at July 29 2020, 12:49:57 PM. (outdated)

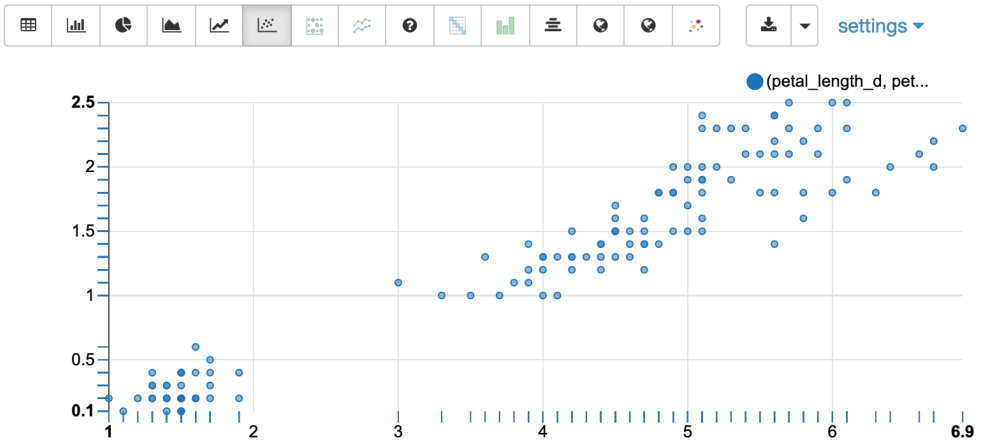
Visualizations

There are a number of visualizations that can flow from the regression result set.

The first visualization shown is a scatter plot with `petal_length_d` on the x-axis and `petal_width_d` on the y-axis. This can be used to visualize the relationship between the two variables in the regression analysis.

```
select linearfit(petal_length_d, petal_width_d) as prediction,
       residual,
       petal_length_d,
       petal_width_d
    from iris
```

FINISHED

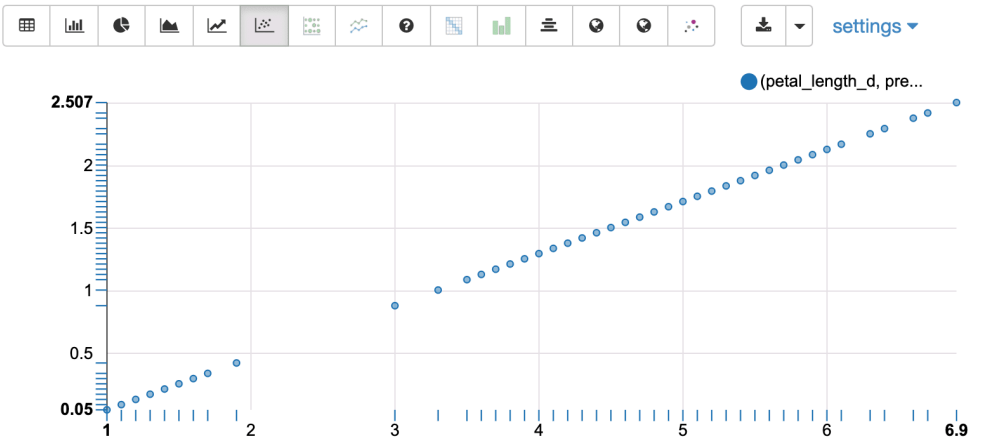


Took 0 sec. Last updated by anonymous at July 29 2020, 12:49:57 PM. (outdated)

The second visualization shows the `petal_length_d` variable on the x-axis and the prediction for `petal_width_d` on y-axis.

```
select linearfit(petal_length_d, petal_width_d) as prediction,
       residual,
       petal_length_d,
       petal_width_d
    from iris
```

FINISHED

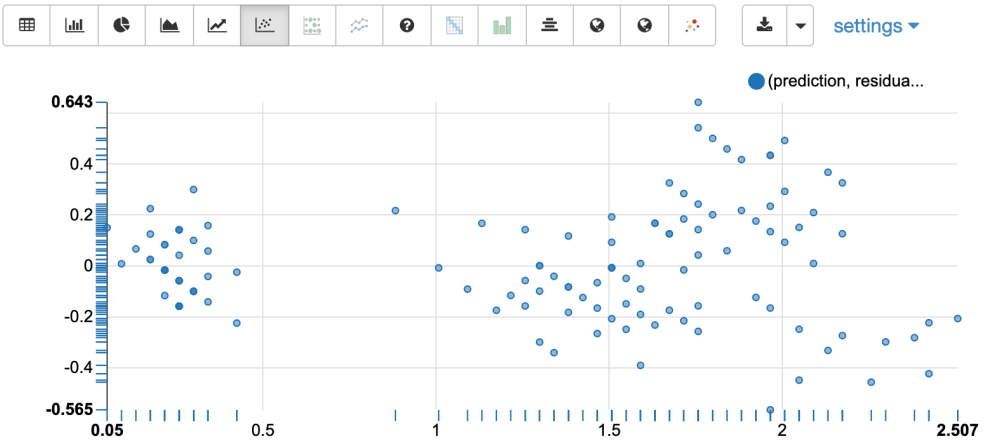


Took 0 sec. Last updated by anonymous at July 29 2020, 12:54:41 PM. (outdated)

The last visualization plots the predictions on the x-axis and the residual on the y-axis. This residual plot can be used to visualize the error of the regression model across the full range of predictions.

```
select linearfit(petal_length_d, petal_width_d) as prediction,
       residual,
       petal_length_d,
       petal_width_d
  from iris
```

FINISHED



Took 0 sec. Last updated by anonymous at July 29 2020, 12:54:41 PM. (outdated)

Significant Terms (significant_terms)

The `significant_terms` function finds anomaly terms in a text or string field that appear more frequently in a search result set than the entire index.

The `significant_terms` function takes four parameters:

1. The string or text field in which to find the terms.
2. The minimum term length to be considered a significant term.
3. The minimum document frequency for a term to be considered a significant term. If greater than one, this value is treated as an absolute number of documents. If the value is a float between 0 and 1, it is considered to be a percentage of total documents.
4. The maximum document frequency for a term to be considered a significant term. If greater than one, this value is treated as an absolute number of documents. If the value is a float between 0 and 1, it is considered to be a percentage of total documents.

Sample syntax

```
select significant_terms(complaint_type_s, 5, 1, .5) as term, foreground,
background, score from nyc311 where borough_s = 'MANHATTAN' limit 10
```

Result set

The result set for the `significant_terms` function contains one row for each significant term. The `significant_terms` function returns the value of the term. There are three additional fields available when the significant_terms function is used:

- The **foreground** field returns the number of documents that contain the term within the result set.
- The **background** field returns the number of documents that contain the term in the entire index.
- The **score** field returns the score for the field which is calculated based on the background and foreground counts. Terms are returned in score descending order.

Sample result set in Apache Zeppelin

```
select significant_terms(complaint_type_s, 5, 1, .5) as term,
       foreground,
       background,
       score
from nyc311
       where borough_s = 'MANHATTAN'
```

FINISHED

settings

term	foreground	background	score
DOF Parking - Tax Exemption	1097	1101	69.62161
Taxi Report	1466	2224	66.319916
Panhandling	480	679	65.892136
Vending	3264	5117	65.15038
Homeless Encampment	3068	4940	65.024345
Found Property	230	295	64.48934
Derelict Bicycle	921	1952	63.62175
Special Projects Inspection Team (SPIT)	1601	3743	62.664524

Took 1 sec. Last updated by anonymous at August 03 2020, 8:56:05 AM. (outdated)

Visualization

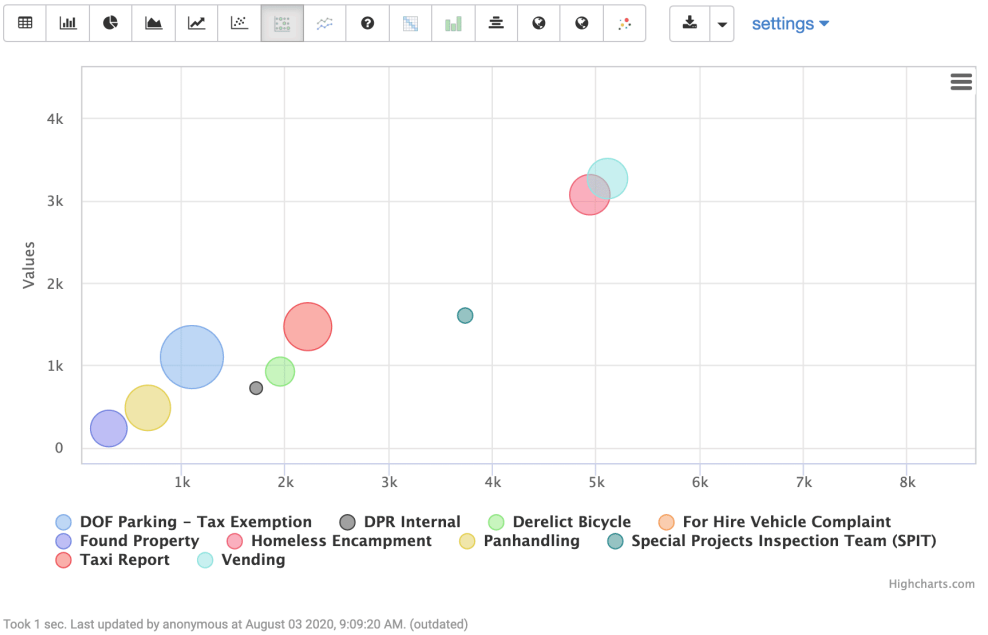
The `significant_terms` result is shown below visualized in an Apache Zeppelin bubble chart. In the bubble chart, the:

- Background counts are plotted on the x-axis
- Foreground counts are plotted on the y-axis
- Bubble size is determined by the score
- Term is displayed in the color coded legend

The bubble chart displays how many documents contain a term, both in the entire index and in the query result set, and how it influences the score.

```
select significant_terms(complaint_type_s, 5, 1, .5) as term,
       foreground,
       background,
       score
from nyc311
       where borough_s = 'MANHATTAN'
limit 10
```

FINISHED



Took 1 sec. Last updated by anonymous at August 03 2020, 9:09:20 AM. (outdated)

Text Analysis and Analytics (analyze)

The `analyze` function analyzes a text or string field with a Lucene/Solr analyzer and returns the analyzed tokens. The use of Lucene/Solr analyzers at query time allows for real-time entity extraction by configuring OpenNLP, Stanford NLP, or other NLP analyzers.

The `analyze` function takes three parameters:

1. The text or string field to analyze
2. A string literal pointer to the Lucene/Solr analyzer to use
   This can be either a field name in the Solr schema with an analyzer defined, or a string that matches a dynamic field pattern.
3. The number of documents to analyze
   Which documents are analyzed is determined by the `WHERE` clause. The documents are ordered by score descending to determine the top N documents to analyze.

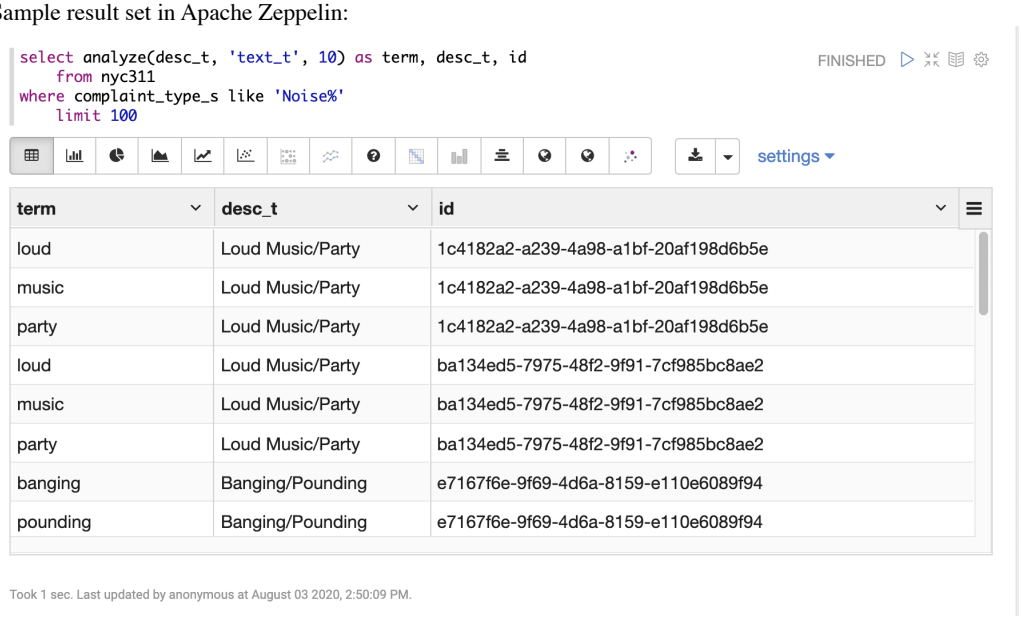
Sample syntax

```
select analyze(desc_t, 'text_t', 10) as term, desc_t, id from nyc311 where
complaint_type_s like 'Noise%' limit 100
```

Result set

The analyzed result set contains a row for each analyzed term. The `analyze` function returns the term. Other fields from the record can be output with the term as well. The number of terms returned in the result set is controlled by the LIMIT clause.

Sample result set in Apache Zeppelin:



Text analytics

A SQL aggregation query can be wrapped around the analyzed result set to perform aggregation over the terms. Below is an example of the wrapped aggregation query:

```
select term, count(*) from (select analyze(desc_t, 'text_t', 10) as term, desc_t,
id from nyc311 where complaint_type_s like 'Noise%' limit 100) group by term order
by count(*) desc
```

Visualization

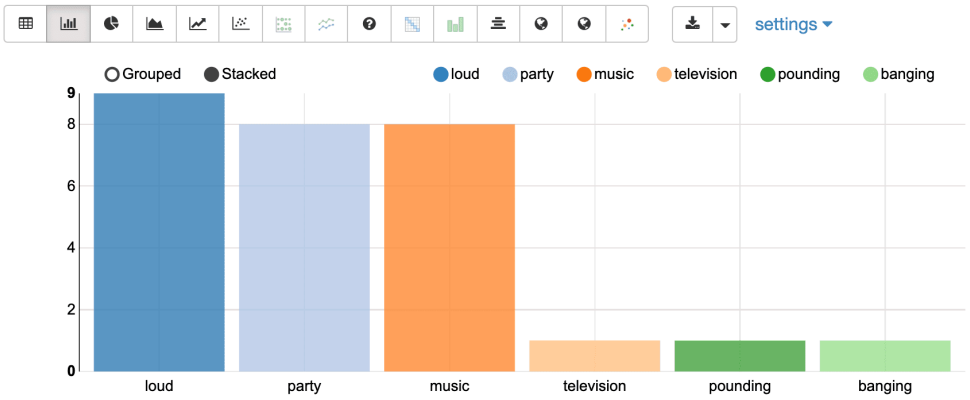
The aggregations over the analyzed result set can be visualized as bar or pie charts. The example below shows the aggregations visualized as a bar chart:

```
select term, count(*) from

    (select analyze(desc_t, 'text_t', 10) as term, desc_t, id
          from nyc311
     where complaint_type_s like 'Noise%'
          limit 100)

group by term
      order by count(*) desc
```

FINISHED



Took 1 sec. Last updated by anonymous at August 03 2020, 3:08:46 PM. (outdated)

Time Series (date_format)

Fusion SQL provides a powerful and flexible time series aggregation query through the use of the `date_format` function. Aggregations that group by a `date_format` are translated to a Solr range facet query. This allows for fast, interactive time series reporting over large data sets.

An example of a time series aggregation is shown below:

```
select date_format(rec_time, 'yyyy-MM') as month, count(*) as cnt from logrecords
where rec_time > '2000-01-01' and rec_time < '2010-01-01' group by month
```

The `date_format` function is used to specify both the output format and the time interval in one compact pattern as specified by the Java SimpleDateFormat *(https://docs.oracle.com/javase/8/docs/api/java/text/SimpleDateFormat.html)* class.

The example above is performing a monthly time series aggregation over the `rec_time` field which is a `datetime` field.

To switch to a daily time series aggregation all that is needed is to change the date pattern:

```
select date_format(rec_time, 'yyyy-MM-dd') as day, count(*) as cnt from logrecords
where rec_time > '2000-01-01' and rec_time < '2000-12-31' group by day
```

Interval Gap

The interval gap can be defined with the optional `gap` parameter. If the gap parameter is not defined it is inferred from the date format string. For example the date format string 'yyyy-MM' will have an inferred gap of 1MONTH.

To specify a custom gap you can add the gap parameter to the date format string as follows:

```
select date_format(rec_time, 'yyyy-MM-dd gap=2DAYS') as day, count(*) as cnt from
logrecords where rec_time > '2000-01-01' and rec_time < '2000-12-31' group by day
```

The time series above will compute a time series with 2 day intervals between buckets.

Interval gaps can be specified for: YEARS, MONTHS, DAYS, HOURS, MINUTES and SECONDS.

Date math predicates

Fusion SQL also supports date math predicates through the `date_add`, `date_sub`, and `current_date` functions.

Below is an example of the use of date math predicates.

```
select date_format(rec_time, 'yyyy-MM-dd gap=3DAYS') as day, count(*) as cnt from
logrecords where rec_time > date_sub(current_date(), 30) group by day
```

Auto-filling of time intervals

Fusion SQL automatically fills any time interval that does not contain data with zeroes. This ensures that the full time range is included in the output which makes the time series results easy to visualize in charts.

Sort Order

Time series aggregations are sorted by default in time ascending order. The `ORDER BY` clause can be used to sort time series aggregation results in a different order.

Having Clause

A `HAVING` clause can also be applied to a time series query to limit the results to rows that meet specific criteria.

click_pos

The `click_pos` UDF computes either a reciprocal rank or a raw click position (using a 0-based index) of a document in a page of results. This UDF is used to compute the mean reciprocal rank (MRR) for experiments. For example, given the following list of documents and a doc ID, the `click_pos` UDF will return 2:

```
docs: a,b,c,d doc ID: c
```

concat_text

The `concat_text` UDF combines multi-valued text fields coming from Solr into a field with a single value delimited by spaces. This UDF is useful when a field returned from Solr uses the `_txt` suffix, which indicates a multi-valued text field.

timediff

The `timediff` UDF computes the difference, in milliseconds, between two timestamps in the same row. From the `session_rollup` job, the `timediff` function computes the difference between the current time and the last activity in a session.

ttest

The `ttest` function performs a two sided t-test on two random samples drawn from two queries. The `ttest` function takes four parameters:

1. The numeric field on which to perform the t-test
2. The query for drawing the first sample
3. The query for drawing the second sample
4. The sample size for both samples

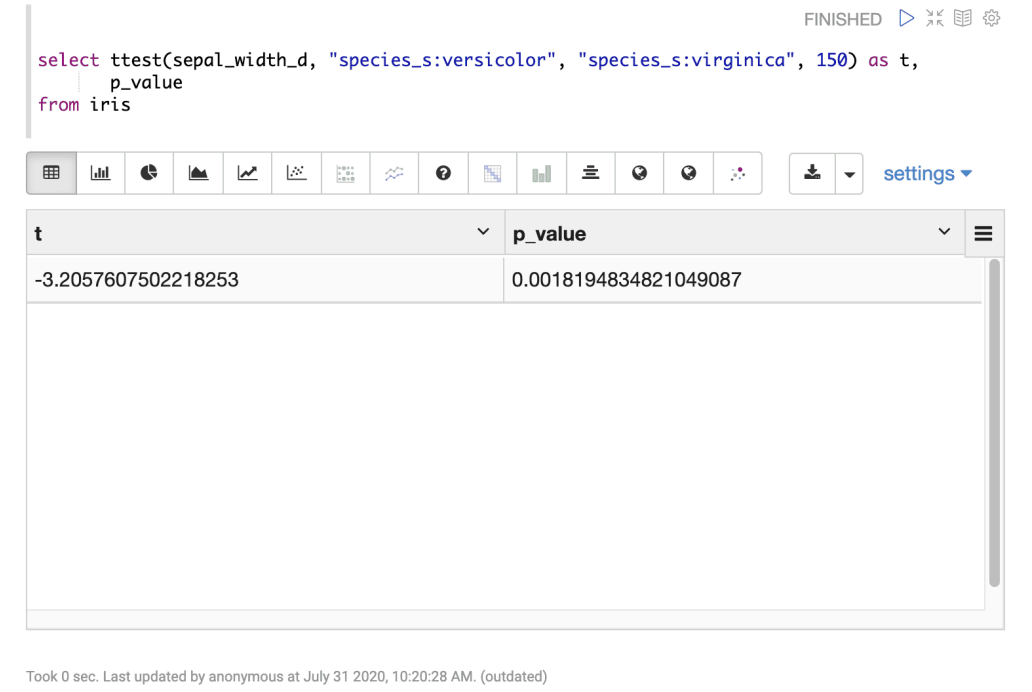
Sample syntax

```
select ttest(sepal_width_d, "species_s:versicolor", "species_s:virginica", 150) as
t, p_value from iris
```

Result set

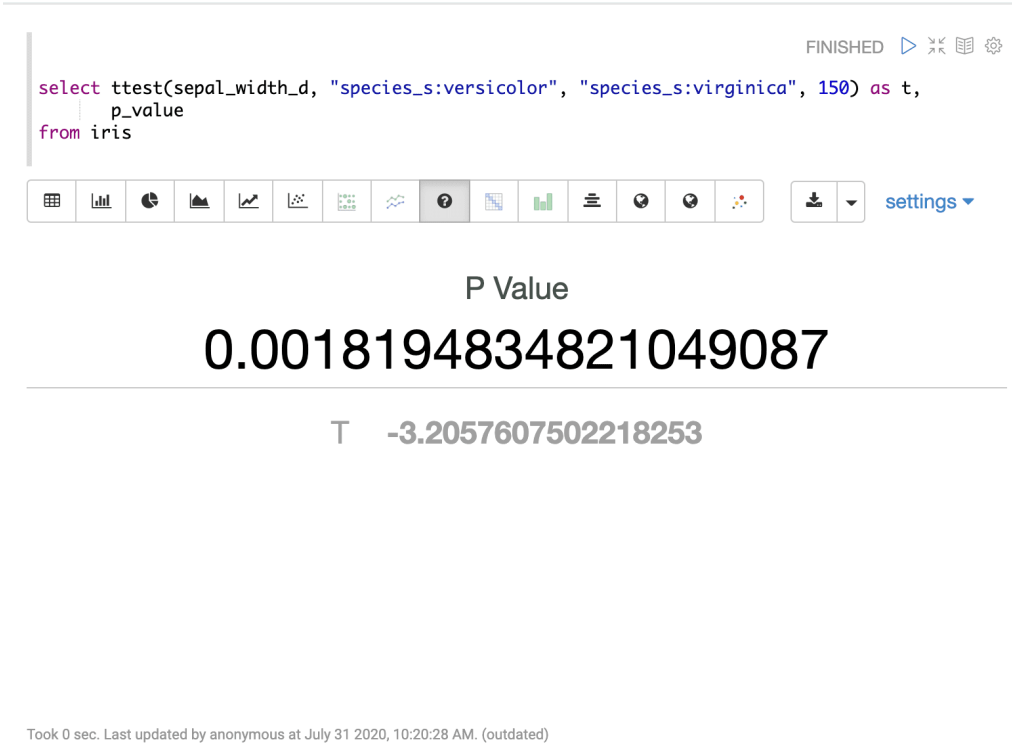
The result set for the ttest function contains one row with the results of the ttest. The ttest function returns the t statistic. The p_value field can be selected to return the p-value for the ttest.

Sample result set in Apache Zeppelin:



Visualization

Sample visualization of the ttest function using the Apache Zeppelin number visualization:



weighted_sum

The `weighted_sum` UDAF takes a weight, type, and type-weight mapping to produce an aggregated weight. For example, consider the following SQL snippet:

```
SELECT query, doc_id, filters, weighted_sum(typed_weight_d, type,
'click:1.0,cart:10.0') AS weight_d FROM signal_type_groups GROUP BY query, doc_id,
filters
```

When applied to the rows in the table below, the `weighted_sum` function produces a final `weight_d` of 12.0 (2*1.0 + 1*10.0). The UDAF is passed rows grouped by `query`, `doc_id`, and `filters`.

query	type	doc_id	filters	typed_weight_d
iPad	click	1	gear	2
iPad	cart	1	gear	1

App Insights

App Insights provides detailed, real-time, searchable reports and visualizations derived from your *signals data*. It also provides alerts and triggers to notify you when specific events occur.

- To open App Insights from the Fusion workspace, navigate to **Analytics > Insights**.
- To switch apps while in App Insights, hover over , and then click a different app.
- To exit App Insights, hover over , and then click **Return to Fusion**.

App Insights pages

When App Insights is open, you can hover over the left navigation panel to reach these pages:

- *Dashboards*
  View graphs and tables about:
  - requests
  - queries
  - results
  - clicks
  - users
  - sessions

  You can filter by timeframe or by the content of the data, and create custom reports.
- *Events*
  View histograms and timelines about events, filtered by:
  - event type
  - users
  - request/response information

  You can also create custom reports about events here.
- *Sessions*
  View charts and timelines about search events, filtered by:
  - user
  - subject
  - event duration
  - events per session
- *Analytics*
  A variety of standard reports are available here. You can also define custom reports to suit your needs.
- *Experiment results*
  If you have configured an *experiment*, you can see visualizations about the results here.

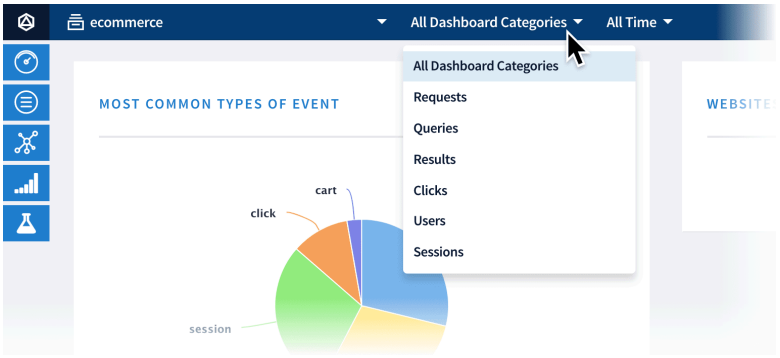
Reports

App Insights provides a standard set of reports, plus the ability to *create custom reports*. Standard reports are located on the *Analytics* page, while custom reports can be defined on the *Dashboard*, *Events*, or Analytics pages. Report data can be filtered by time or by free text search, and reports can be exported by clicking

Export table.

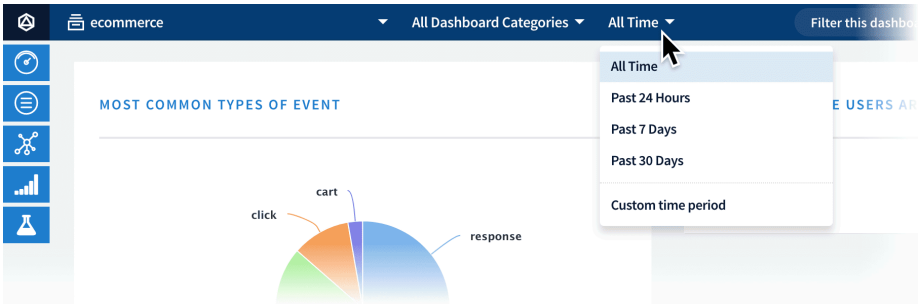
Dashboards in App Insights

When you open the Dashboards page, it displays graphs and tables for attributes that are common to all signals. For more specific charts, you can open the **All Dashboard Categories** menu at the top and select a category:

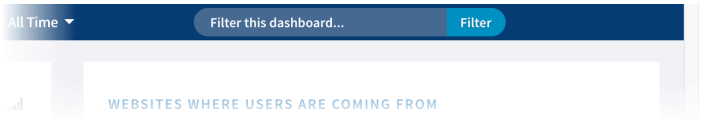


Dashboard filtering

You can filter by timeframe using the **All Time** menu:



To filter by the content of the data, enter a string in the **Filter this dashboard…** text box:

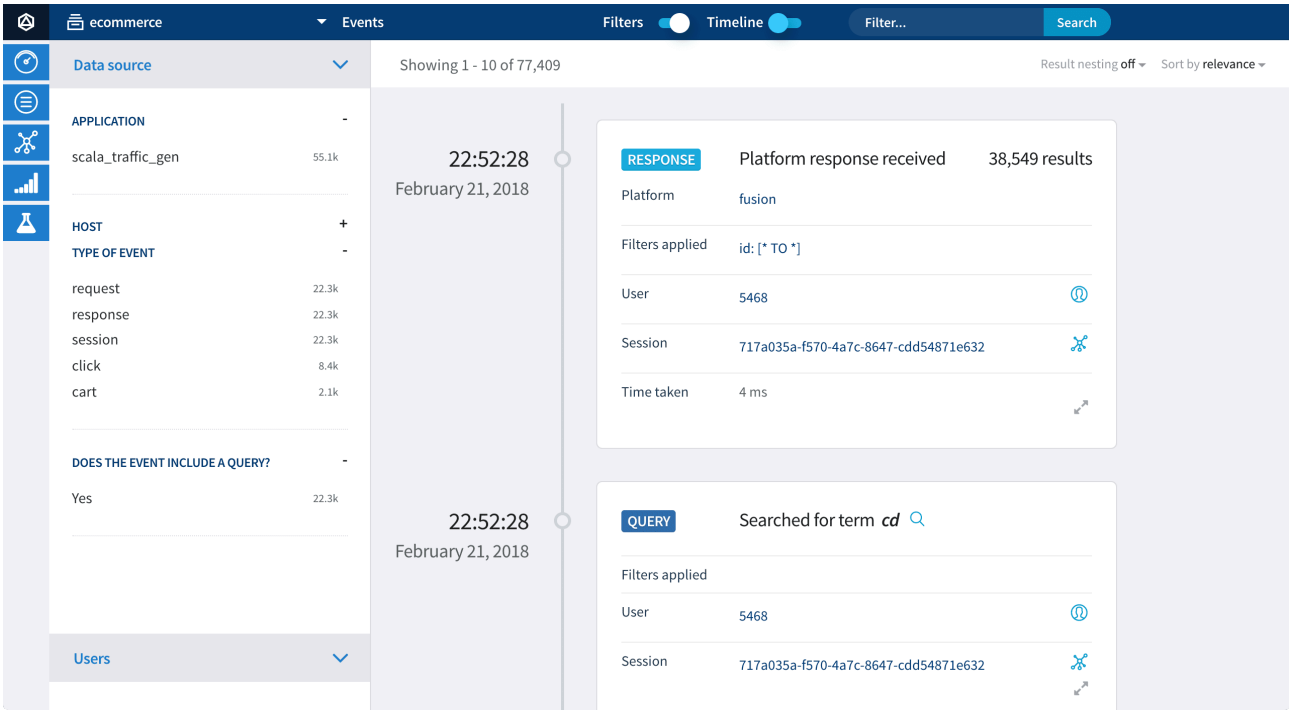


For more information, see *dashboards*.

Events

Click **Events** to view histograms and timelines about events, filtered by:

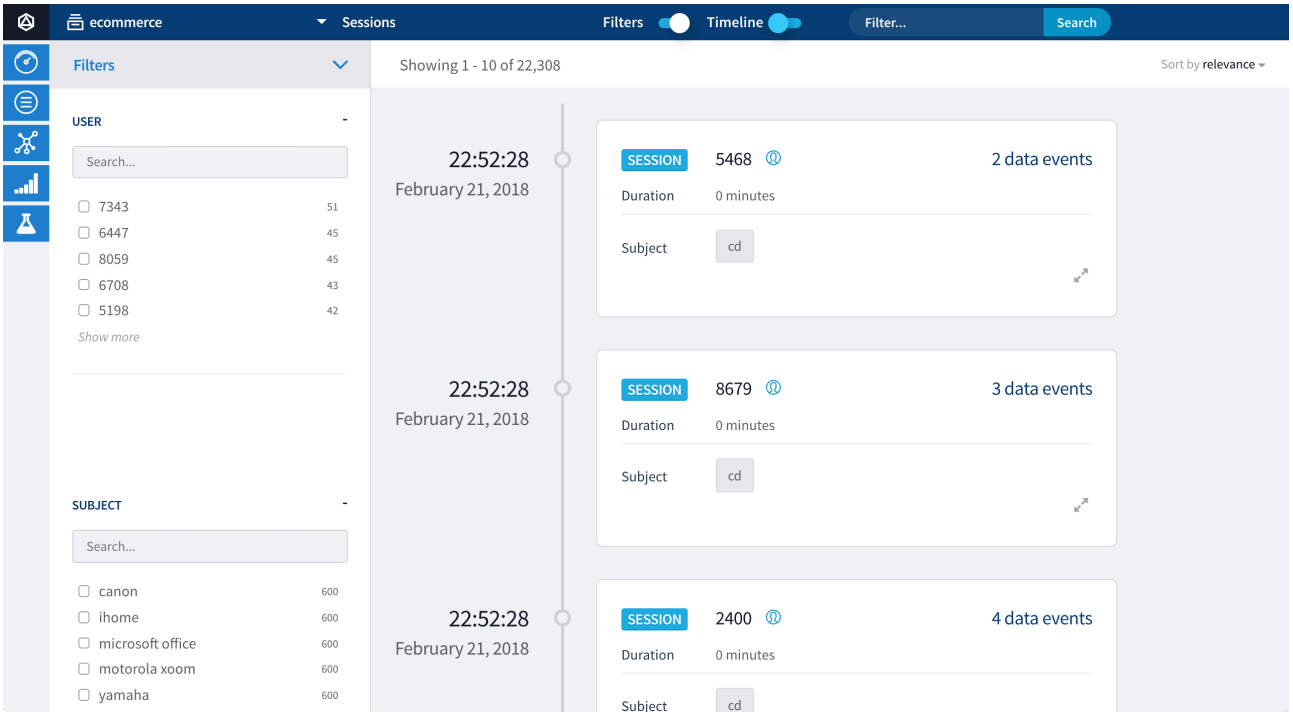
- event type
- users
- request/response information



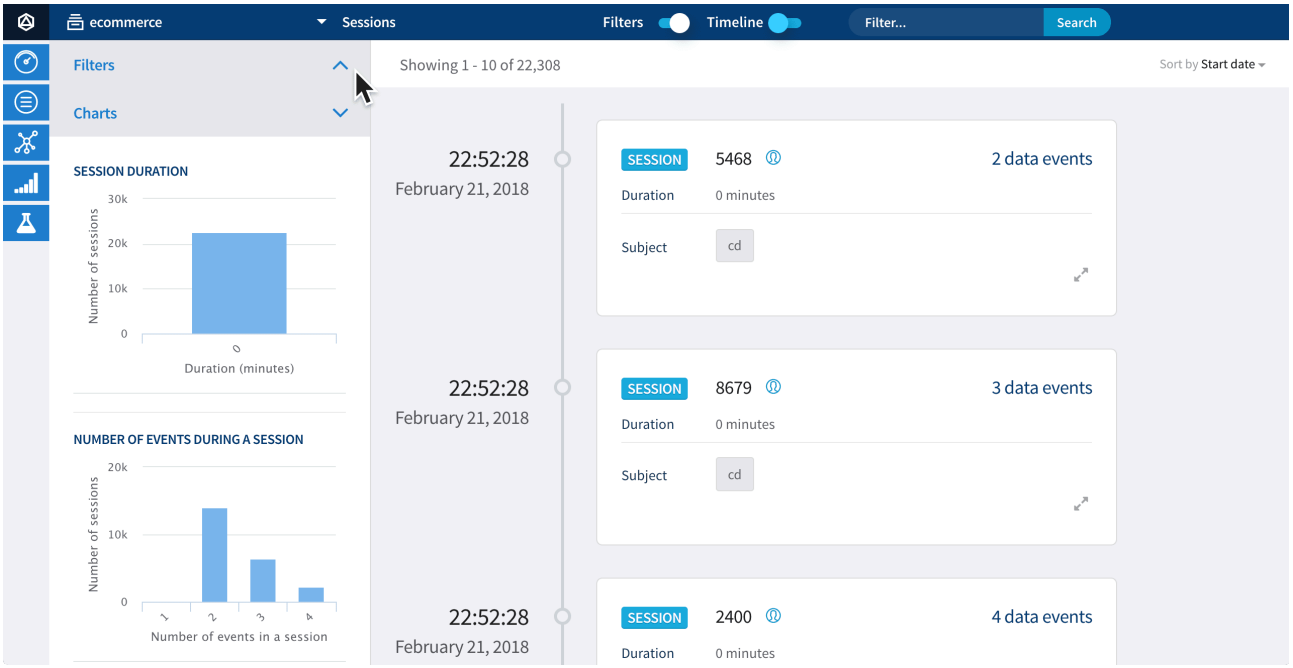
Sessions

Click **Sessions** to view charts and timelines about search events, filtered by:

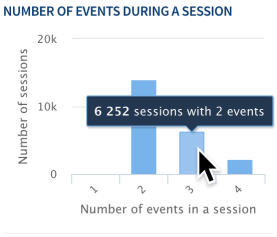
- user
- subject
- event duration
- events per session



Charts are located on the left, below the filters. Scroll down or minimize the filters to view the charts:

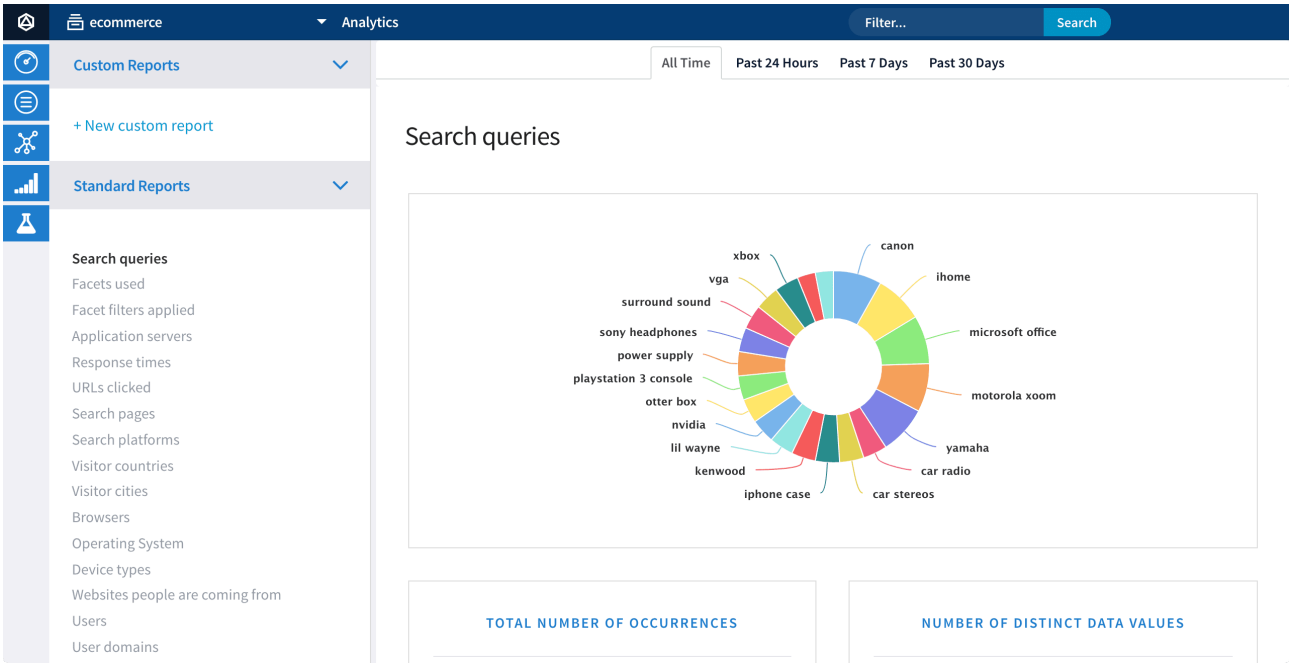


Hover over the bars on any chart to filter the timeline by event duration or events per session:



Analytics

Analytics include a standard set of reports, plus the ability to create custom reports. Custom reports can also be defined on the *Dashboards* and *Events* pages. Report data can be filtered by time or by free text search, and reports for tabular data can be exported by clicking **Export table**.



Standard reports

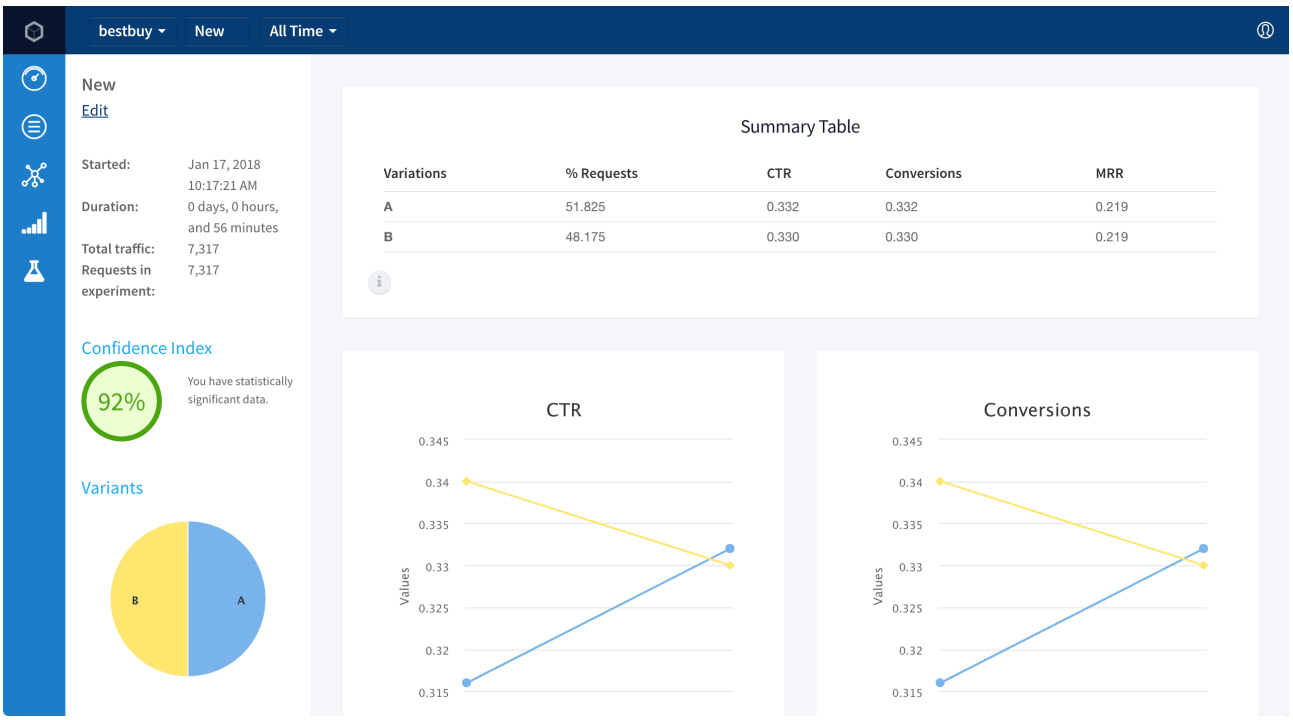
To view the standard reports, click **Analytics**, then select one of the standard reports:

- Facets used
- Facet filters applied
- Application servers
- Applications
- Response times
- URLs clicked
- Type of query
- Search pages
- Search platforms
- Types of response
- Visitor countries
- Visitor cities
- Browsers
- Operating System
- Device types
- Websites people are coming from
- Users
- User domains
- Types of event
- Head Tail analysis

Experiment Results

If you have configured an *experiment*, you can see visualizations about the results here.

This example shows a 92% confidence index for the difference in click-through rate (CTR). Conversions use the same signal type (`click`), so the result is identical.



Requirements for Reports and Dashboards

Although there are many fields that are relevant to *signals*, this topic describes which fields are **required** for each *type of signal*. Without these fields, reports and dashboards may not function as expected.

Report	Signal Types Used	Required Field	Type	Description	Example
Search queries	✘ Annotation ✅ Click ✘ Login ✅ Request ✅ Response	`query`	string	Terms of the query.	`physics`
Facets used	✘ Annotation ✘ Click ✘ Login ✅ Request ✘ Response	`filter_field`	array of string	List of filters associated with the query, which in turn is associated with signal.	`["categories_s"]`
Facet filters applied	✘ Annotation ✘ Click ✘ Login ✅ Request ✘ Response	`filter`	array of string	List of **applied** filters associated with the query, which in turn is associated with signal.	`["categories_s/math.OA"]`
Application servers	✘ Annotation ✅ Click ✘ Login ✅ Request ✘ Response	`host`	string	Host name of the server that is hosting the app, which in turn is generating the signal.	`lucidworks`
Response times	✘ Annotation	`time`	number	Response time (in milliseconds).	`17`

Report	Signal Types Used	Required Field	Type	Description	Example
	✘ Click ✘ Login ✘ Request ✅ Response				
URLs clicked	✘ Annotation ✅ Click ✘ Login ✘ Request ✘ Response	`url`	string	URL of the page that was selected from the search results. This typically results from a click signal.	`http://arxiv.org/pdf/astro-ph/0611688v1`
Search pages	✘ Annotation ✘ Click ✘ Login ✅ Request ✘ Response	`page_title`	string	Title of the page that the signal originated from.	`Search Preview`
Search platforms	✘ Annotation ✅ Click ✘ Login ✅ Request ✅ Response	`platform`	string	Name of the search platform.	`fusion`
Visitor countries	✘ Annotation ✅ Click ✘ Login	`geo_country_name`	string	Originating country of the user.	`United States`

Report	Signal Types Used	Required Field	Type	Description	Example
	✅ Request ✘ Response				
Visitor cities	✘ Annotation ✅ Click ✘ Login ✅ Request ✘ Response	`geo_city_name`	string	Originating city of the user.	`San Francisco`
Browsers	✘ Annotation ✅ Click ✘ Login ✅ Request ✘ Response	`browser_name`	string	Name of the browser that generated the signal.	`Firefox`
Operating System	✘ Annotation ✅ Click ✘ Login ✅ Request ✘ Response	`os_name`	string	Name of the operating system used by the user.	`Ubuntu`
Device types	✘ Annotation ✅ Click ✘ Login ✅ Request	`os_device`	string	Type of the device used by the user.	`Computer`

Report	Signal Types Used	Required Field	Type	Description	Example
	✘ Response				
Websites people are coming from	✘ Annotation ✅ Click ✘ Login ✘ Request ✘ Response	`referrer_domain`	string	URL of the page that the signal originated from.	`http://www.google.com`
Users	✘ Annotation ✅ Click ✘ Login ✅ Request ✘ Response	`user_id`	string	Unique ID for the user that generated the signal.	`admin`
User domains	N/A	`domain`	string	Domain of the user that generated the signal.	`lucidworks.com`
Types of event	✅ Annotation ✅ Click ✅ Login ✅ Request ✅ Response	`type`	string	Type of the signal.	`click`
Head Tail analysis	✘ Annotation			This report is populated by running a head tail job in Fusion. With a collection called `foo`, for example, run the `foo_head_tail` job.	N/A
	✅ Click	type	string		N/A
	✅ Click	query	string		N/A

Report	Signal Types Used	Required Field	Type	Description	Example
	✘ Login				N/A
	✘ Request				N/A
	✅ Response	count_i	number		N/A

The fields mentioned above are defined as follows:

Field Name	Example Value	Description
`count_i`	1	Number of times an interaction event occurred with this item
`doc_id`	NMDDV	Product ID or Item ID
`query`	xwearabletech	A query string from the user
`session_id`	91aa66d11af44b6c90ccef44d055cf9a	Id for session in which user generated the signal
`type`	quick_view_click	Type of session the user used to interact with the platform
`user_id`	11506893	ID of user during the session
`timestamp_tdt`	2018-11-20T17:58:57.650Z	Time when signal was generated

Some *signal types*, including *custom signal types*, may include additional fields.

Experiments

When making changes to a query pipeline or query parameters that will affect users' search experience, it is often a good idea to run an experiment in order to verify that the results are what you intended. Fusion lets you create and run experiments that take care of dividing traffic between variants and calculating the results of each variant with respect to configurable objectives such as purchases, click-through rate, or search relevance.

There are two ways that a search application might interact with an experiment:

- Preferred: Using a query profile
- Alternative: Using an Experiment query pipeline stage

If a query profile is configured to use an experiment, then a search app sends queries and signals to the query profile endpoint. If the experiment is active, then Fusion routes each query through one of the experiment variants. The search app will also send subsequent signal data relating to that query — clicks, purchases, "likes", or whatever is relevant to the application — to that same query profile, and Fusion will record it along with information about the experiment variant that the user was exposed to.

Fusion generates and stores the data that metrics calculations use. It also automatically creates jobs that periodically calculate the metrics. After metrics have been calculated, they are available in *App Insights*.

This topic explains the experiment workflow and basic concepts. These additional topics provide details about how to implement experiments to improve the user experience:

- *Plan an Experiment*
- *Set Up an Experiment*
- *Run an Experiment*
- *Analyze Experiment Results*

Run an Experiment Tutorial

The *Run an Experiment* tutorial takes you through the steps needed to run an A/B experiment to compare metrics such as click-through rate (CTR) and query relevance for two differently configured query pipelines. You plan the experiment, create a Fusion app, index a datasource, and create a query profile that includes the configuration data needed for experiments. In Fusion, you start and stop the experiment. A search app uses the query profile for Fusion queries. Different users get different search results, but they are blissfully unaware that an experiment is going on.

A/B/n experiments

Fusion's experiments feature set implements A/B/n experiments *(https://en.wikipedia.org/wiki/A/B_testing)*, also called A/B experiments or A/B tests, where A and B are *experiment groups* with one or more *variants*.

Fusion's implementation of an A/B experiment uses consistent hashing on a unique ID field (typically `userId`), concatenated with the experiment's name, to assign each request to one of the experiment groups. Any future requests with that hash are assigned to the same group, guaranteeing user "stickiness".

If you prefer "stickiness" only at the session level, you can send a session ID instead of a user ID.
If you send no ID at all, the request is not assigned to a variant since there is no way to consistently assign it to the same one. In that case, the request uses the "default" configuration of the query profile or experiment stage.

Example

The following experiment is an example of an A/B/n experiment with three variants:

- **Variant 1 (control).** Use the default query pipeline with no modifications. Each experiment should have a "control" variant as the first variant; the other variants will be compared against this one.
- **Variant 2 (content-based filtering with a Solr MoreLikeThis stage).** Content-based filtering uses data about a user's search results, browsing history, and/or purchase history to determine which content to serve to the user. The filtering is non-collaborative.
- **Variant 3 (collaborative filtering with a Recommend Items for User stage).** Collaborative filtering takes advantage of knowledge about the behavior of many individuals. It makes serendipitous discovery possible—a user is presented with items that other users deem relevant, for example, socks when buying shoes.

High-level workflow

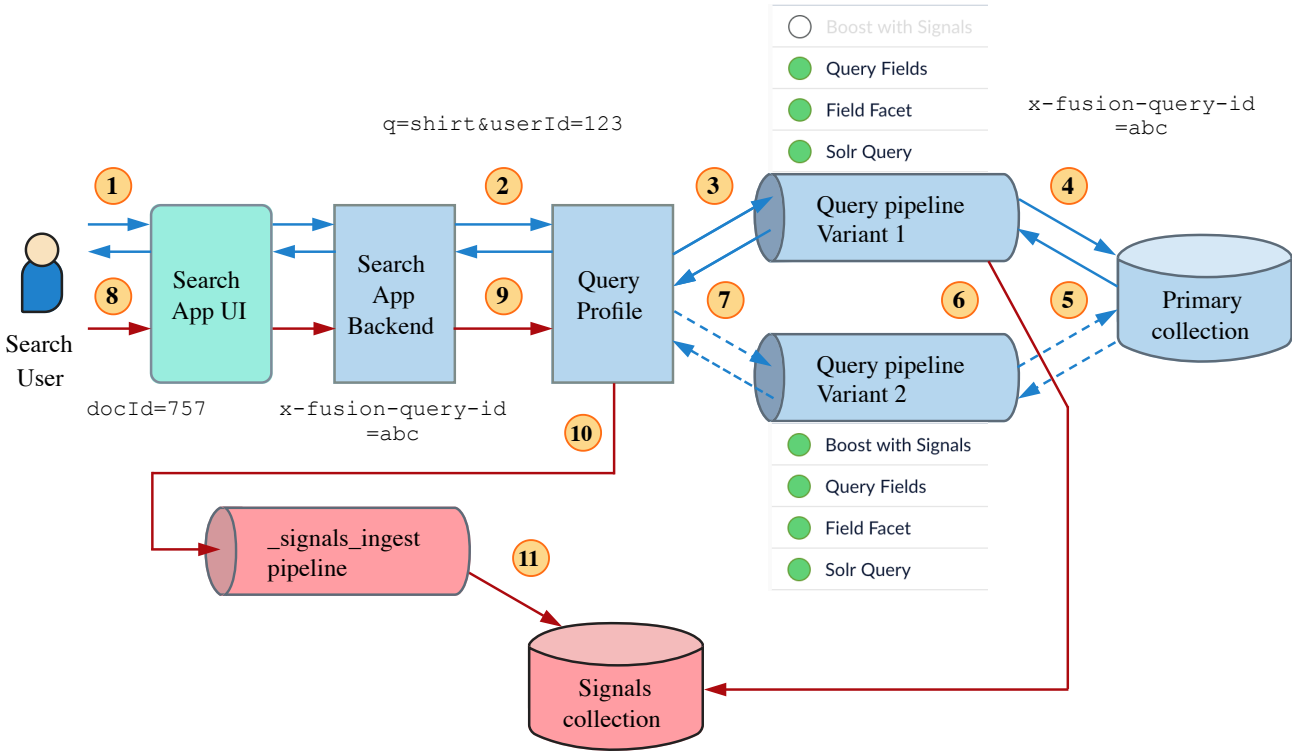
In an experiment:

1. A Fusion administrator defines the experiment. An experiment has *variants* with differences in query pipelines, query pipeline stages, collections, and/or query parameters.
2. The Fusion administrator assigns the experiment to a query profile.

3. A user searches using that query profile.
4. If the experiment is running, Fusion assigns the user to one of the experiment variants, in accordance with *traffic weights*. Assignment to a variant is persistent. The next time the user searches, Fusion assigns the same variant.
5. Different experiment variants return different search results to users.
6. Users interact with the search results, for example, viewing them, possibly clicking on specific results, possibly buying things, and so forth.
7. Based on the interactions, the search app backend sends signals to the signals endpoint of the query profile for the experiment.
8. Using signal data, an automatically-created Spark job periodically computes metrics for each experiment variant and writes the metrics to the `job_reports` collection.
9. In the Fusion UI, an administrator can view reports about the experiment.
10. Once the results of the experiment are conclusive, the Fusion administrator can stop the experiment and change the query profile to use the winning variant, or start a new experiment.

Information flow

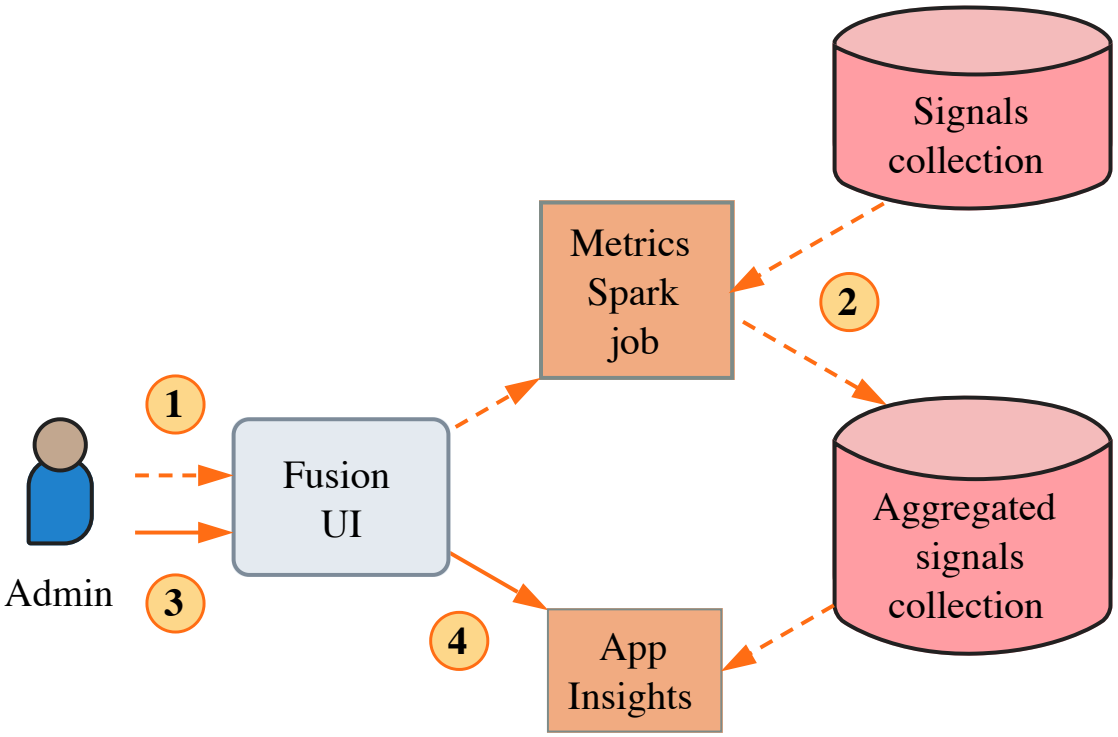
This diagram illustrates information flow through an experiment. Numbers correspond to explanations below the diagram.



1. A user searches in a search app. For example, the user might search for `shirt`.
2. The search app backend appends a `userId` or other unique ID that identifies the user, for example, `userId=123`, to the query and sends the query to the query profile endpoint for the experiment.
3. Using information in the query profile and the value of the unique ID, Fusion routes the query through one of the experiment's variants. In this example, Fusion routes the query through query pipeline 1.
4. A query pipeline adds a `x-fusion-query-id` to the response header, for example, `x-fusion-query-id=abc`.
5. Based on the query, Fusion obtains a search result from the index, which is stored in the primary collection. Fusion sends the search result back to the search app.
6. Fusion sends a response signal to the signals collection.
7. A different user might be routed through the other experiment variant shown here, and through query pipeline 2. This query pipeline has an enabled Boost with Signals stage, unlike query pipeline 1.
8. The search user interacts with the search results, viewing them, possibly clicking on specific results, possibly buying things, and so forth. For example, the user might click the document with `docId=757`.
9. Based on the interactions, the search app backend sends click signals to the signals endpoint for the query profile. Signals include the same query ID so Fusion can associate the signals with the experiment. Specifically, the click signal *must* include a field named `fusion_query_id` in the `params` object of the raw click signal whose value was returned in the response object in a header named `x-fusion-query-id`. If you are tracking queries and responses with App Studio, the `fusion_query_id` parameter will be passed with the click signal as long as you specify the appropriate response attribute in your `track:clicks` tag.
10. Using information in the query profile, Fusion routes the signals to the `_signals_ingest` pipeline.
11. The `_signals_ingest` pipeline stores signals in the `_signals` collection. Signals include the collection ID of the primary collection and experiment tracking information.

Metrics generation

This diagram illustrates metrics generation:



1. A Fusion administrator can configure which metrics are relevant for a given experiment and the frequency with which experiment metrics are generated. They can also generate metrics on demand.
2. Using signal data, an automatically-created Spark job periodically runs in the background. It obtains signal data from the `COLLECTION_NAME_signals` collection, computes metrics for each experiment variant, and writes the metrics to the collection used for aggregated signals (`_signals_aggr`).
3. In the Fusion UI, a Fusion administrator can view experiment metrics.
4. These calculated metrics are used to display reports about the experiment.

Experiment Metrics

This section describes metrics available for experiments.

Click-Through Rate

The Click-Through Rate (CTR) metric provides the rate of clicks per query for a variant. The CTR is a number between 0 and 1, that is, what proportion of queries lead to clicks. Variants with a CTR closer to 1 perform better than variants with a lower rate.

CTR is *cumulative*, that is, each time it is calculated, it is calculated from the beginning of the experiment. After each variant has reached a stable level, you should not see large day-to-day fluctuations in the CTR.

The automatically-created job that generates the Click-Through Rate metrics is named `<experiment-name>-<metric-name>`, for example, `Experiment-CTR`.

Conversion Rate

The Conversion Rate metric provides the rate of some type of signal per variant, that is, what proportion of queries lead to some type of signal, such as `cart`, `purchase` or `like` signals. (These signal types are not predefined.)

For example, if you are interested in how many queries convert into `cart` signals, specify the `cart` signal type in the conversion rate metric.

The Click-Through Rate metric is a conversion rate for `click` signals.

The automatically-created job that generates the Conversion Rate metrics is named `<experiment-name>-<metric-name>`, for example, `Experiment-Conversion`.

Mean Reciprocal Rank (MRR)

The Mean Reciprocal Rank (MRR) metric measures the position of documents that were clicked on in ranked results. It ranges from 0 (at the very bottom) to 1 (at the very top). MRR penalizes clicks that occur further down in the results, which indicate a ranking issue where relevant documents are not ranked high enough. Variants with an MRR closer to 1 indicate that users are clicking on documents that have higher ranks.

The automatically-created job that generates the Mean Reciprocal Rank metrics is named `<experiment-name>-<metric-name>`, for example, `Experiment-MRR`.

Response Time

The Response Time metric computes the named statistic (for example, `mean`, `variance` or `max`) from response-time data. The default statistic is `avg` (average, the same as `mean`).

You can use the Response Time metric to evaluate the impact of adding additional stages to a query pipeline, for example, a recommendation or machine learning stage.

The response time is the end-to-end processing time from when a query pipeline receives a query to when the pipeline supplies a response:

- **No Experiment stage.** If a query pipeline does not have an Experiment stage, then there is no experiment-processing overhead in the response times.
- **Experiment stage.** If a query pipeline includes an Experiment stage, then processing by that stage is included in the response times.

The automatically-created job that generates the Response Time metrics is named `<experiment-name>-<metric-name>`, for example, `Experiment-Response_time`.

Supported functions

When adding the Response Time metric to an experiment, specify one of these Spark SQL function names or aliases for the **Statistic**.

Function name or alias	Description
`avg`	Mean response time
`kurtosis`	Kurtosis of the response times
`max`	Maximum response time
`mean`	Mean response time
`median`	Median response time. This is an alias for `percentile(query_time,0.5)`.
`min`	Minimum response time
`percentile_N`	Nth percentile of the response times, that is, the value at or closest to the percentile. `N` is an integer between 1 and 100. This is an alias for the function `percentile(query_time,N/100)`.
`skewness`	Skewness of the response times
`sum`	Sum of the response times
`stddev`	Standard deviation of the response times
`variance`	Variance of the response times

For more information about these functions, see the documentation for Spark SQL Built-in Functions *(https://spark.apache.org/docs/latest/api/sql/index.html)*.

Custom SQL

Under the covers, Fusion computes all experiment metrics using Fusion's *SQL aggregation* engine.

The Custom SQL metric lets you define your own SQL to compute a metric per variant. The SQL must project these three columns in the final output, and perform a GROUP BY on `variant_id`:

- `value`.* A double field that represents the metric provided by this custom SQL
- `count`.* The number of rows used to compute the value for a variant, that is, how many signals contributed to this value
- `variant_id`. The unique identifier of the variant

An internal view named `variant_queries` is built into the experiment job framework. This view is transient and is not defined in the table catalog; it only exists for the duration of the metrics job. The `variant_queries` view exposes all response signals for a given variant ID. The `variant_queries` view exposes the following fields pulled from response signals:

Field	Description
`id`	Response signal ID set by a query pipeline and returned to the client application using the `x-fusion-query-id` response header
`variant_id`	Experiment variant this response signal is associated with
`query_doc_ids`	Comma-delimited list of document IDs returned in the response, in ranked order
`query_timestamp`	ISO-8601 timestamp for the time when Fusion executed the query
`query_user_id`	User associated with the query. The front-end application must supply this.
`query_rows`	Number of rows returned for this query, that is, the page size
`query_hits`	Total number of documents that match this query, that is, the number of documents that were found
`query_offset`	Page offset
`query_time`	Total time to execute the query (in milliseconds)

You can use the `fusion_query_id` field to join the `variant_signals` view with other signal types such as `click`. For example, if you want to get a count of clicks per variant, you would use:

```
1:     SELECT COUNT(1) AS value, COUNT(1) AS count, vq.variant_id as variant_id
2:       FROM ${inputCollection} c
3: INNER JOIN variant_queries vq ON c.fusion_query_id = vq.id
4:      WHERE c.type = 'click'
5:   GROUP BY variant_id
```

In this SQL:

- At line 1, we project the required `value`, `count`, and `variant_id` columns as the output for our custom SQL; this is required for all custom SQL metrics.
- At line 2, we use a built-in macro that represents the input collection for our metrics job. The SQL engine replaces the `${inputCollection}` variable with the correct collection name at runtime, which is typically a signals collection.
- At line 3, we use the `fusion_query_id` column to join `click` signals with the `id` column of the `variant_queries` view. This illustrates how the `variant_queries` view helps simplify the SQL you have to write to build a custom metric.
- At line 4, we filter signals to only include `click` signals. Behind the scenes, Fusion will send a query to Solr with `fq=type:click`.
- At line 5, we group by the variant_id to compute the aggregated metrics for each variant; all Custom SQL must perform a group by variant_id.

To illustrate the power of Custom SQL metrics for experiments, let us build the SQL to compute the average page depth of clicks for each variant, to indicate if users are having to navigate beyond the first page to find results. The intuition behind this metric is that variants having a higher average page depth might indicate a ranking problem. Users are not finding relevant documents on the first page of results.

Specifically, to build our query, we need the `query_offset` and `query_rows` columns associated with each click in a variant:

```
    SELECT AVG((vq.query_offset/vq.query_rows)+1) as value,
           COUNT(1) as count,
           vq.variant_id as variant_id
      FROM ${inputCollection} c
INNER JOIN variant_queries vq ON c.fusion_query_id = vq.id
     WHERE c.type = 'click'
  GROUP BY variant_id
```

In practice, MRR is a better metric for determining the ranked position of clicks, but this SQL gives a basic illustration of how to build Custom SQL metrics.

Lastly, when building Custom SQL metrics, you have the full power of Spark SQL functions, see: https://spark.apache.org/docs/latest/api/scala/index.html#org.apache.spark.sql.functions$ *(https://spark.apache.org/docs/latest/api/scala/index.html#org.apache.spark.sql.functions$)*.

The automatically-created job that generates the Custom SQL metrics is named `<experiment-name>-<metric-name>`, for example, `Experiment-SQL`.

Query Relevance

The Query Relevance metric calculates the performance of queries against a "gold standard" or "ground truth" dataset that lists which documents should be returned for each query. You can either predetermine the queries that will be used and the documents that should be returned, and place them in a Solr collection in the correct format, or let the `groundTruth` job use historical click signals to generate the ground truth data automatically.

Note that the Query Relevance metric *does not* calculate metrics based on live traffic. Instead, it issues the queries specified in the ground truth collection against each variant, and calculates the performance of the queries.

The jobs that generate the Query Relevance metrics are named `<experiment-name>-groundTruth-<metric-name>` and `<experiment-name>-rankingMetrics-<metric-name>`, for example, `Experiment-groundTruth-QR` and `Experiment-rankingMetrics-QR`.

	You must run the `groundTruth` job by hand the first time. Query Relevance `rankingMetrics` jobs that run before the `groundTruth` job runs do not produce metrics. Subsequently, the `groundTruth` job runs once a month.

Ground Truth Queries

Query relevance metrics rely on having a set of queries and a list of documents that should be returned for those queries in ranked order. Specifically, a ground truth dataset contains tuples of query + document ID + weight, such as the following data for a fictitious Home Improvement search application:

Query	Document ID	Weight
hammer	123	0.9
hammer	456	0.8
hammer	789	0.7
masking tape	234	0.85
masking tape	567	0.82
masking tape	890	0.76

Typically, the queries included in the ground truth set represent important queries for a given search application. The weight assigned to each document is used to determine the expected ranking order for the query. Ideally, your ground truth dataset should specify the same number of documents per query, for example. 10. But this is not required technically for computing query relevance metrics. In other words, one query can have 10 documents specified and another query can only specify 5.

In Fusion, you can either load a curated ground truth dataset into a Fusion collection or use Fusion's ground truth job to build a ground truth dataset using signals. If you use the ground truth job, Fusion looks at click/skip behavior for documents by analyzing response and click signals. It follows that you need a sufficient number of signals to generate an accurate ground truth dataset.

The basic intuition behind the ground truth job is that for queries that occur frequently in your search application, whether a user clicks or skips over a document serves as a relevance judgement of a document for a given query. With a sufficient sample size per query, Fusion can decide which documents are relevant and which are not for any given query. It is important to note, however, that, because the ground truth dataset is generated from your click signals, if you have relevant documents that are never clicked (maybe because they are on the second page of results), then they will never appear in your ground truth set.

Calculating Performance vs. Ground Truth

After you have a ground truth dataset loaded into Fusion, the Query Relevance metric will calculate all of the following metrics:

Precision

Precision is the fraction of returned documents that are relevant to the query (that is, how many of the documents returned by this variant exist in the ground truth dataset).

Recall

Recall is the fraction of total relevant docs that are returned by this query (that is, how many of the documents in the ground truth set appear in the result set for this variant).

Normalized Discounted Cumulative Gain (nDGC)

The Normalized Discounted Cumulative Gain (nDCG) indicates whether a variant is returning highly relevant documents near the top of results.

The nDCG has a value between 0 and 1. Larger values indicate that more highly relevant documents occur earlier in the results for a query. Conversely, if a variant returns highly relevant documents lower in the results, then its nDCG score will be lower, penalizing the ranking strategy used by the variant for returning highly relevant documents lower in the results. For more details on nDCG, see https://en.wikipedia.org/wiki/Discounted_cumulative_gain *(https://en.wikipedia.org/wiki/Discounted_cumulative_gain)*.

F1

The F1 score is the harmonic mean between precision and recall at a given depth (10 by default). The F1 score ranges between 0 and 1, with larger values indicating that a variant is achieving a better balance of precision and recall than variants with lower F1 scores. For more details, see https://en.wikipedia.org/wiki/F1_score *(https://en.wikipedia.org/wiki/F1_score)*.

Mean Average Precision (MAP)

The Mean Average Precision (MAP) metric indicates how many documents returned for a query, down to a specific depth, are considered relevant to a query averaged over all queries in the ground truth dataset. MAP is a value between 0 and 1. Larger values mean that the variant returns more relevant than non-relevant documents. For example, if the relevance judgement for a result set containing 3 documents is: 1, 0, 1, then the average precision for that query will be 1/1, 0, ⅔ ~ 0.834 (1.667/2).

Jupyter Support in Fusion

Jupyter *(https://jupyter.org/)* is an open-source web application that allows you to create and share documents that contain live code, equations, visualizations, and narrative text.

Fusion 5 provides a Jupyter service that can be run from the Fusion Helm chart. See *How to enable Jupyter* and *How to use Jupyter with Fusion SQL*.

This feature is not available in Managed Fusion environments.

Why Jupyter

Jupyter is evolving, and it has a number of kernels that support different programming languages. With Jupyter, you can run Spark in Scala or Python and run Fusion SQL at the same time in Python. This versatility gives you options to debug and try out various Fusion features. By integrating BeakerX *(http://beakerx.com/)*, we get access to a wide variety of kernels and visualization features. And by configuring Jupyter, we can hide it behind the Fusion gateway (proxy) service and avoiding exposing the Jupyter IP externally.

What this service is for

- Run/Debug Spark code in Scala/Python (Replacement for spark-shell)
- Run SQL queries via Fusion SQL
- Debug Scala and SQL transforms in PBL jobs
- Everything else for which Jupyter is designed

What this service is not for

This service is not for running Spark jobs in a Kubernetes cluster.

The Jupyter pod does not have access to create or delete pods in a Kubernetes cluster, and therefore you cannot run jobs from Jupyter notebook in a Kubernetes cluster. However, Spark local mode can be used with a higher driver memory if needed.

We recommend sampling data for debugging Jupyter and then running the actual job using the *Fusion jobs UI*.

Configuration tips

- The Jupyter pod IP is `ClusterIP` and not exposed externally. We recommend accessing it via the Fusion proxy and not exposing the pod IP via `LoadBalancer`.
- Jupyter can also be accessed via port-forwarding and available via `localhost:8888/jupyter`
- Idle notebooks and kernels are shutdown after 60 mins of inactivity. Note that any additional libraries installed via `pip` will need to be re-installed when the pod is recycled.
- This image is baked with Python 3.6+.

JupyterHub

Customers who run JupyterHub can use `lucidworks/fusion-jupyter` *(https://github.com/lucidworks/fusion-jupyter)* to launch notebooks with the Fusion Jupyter image.

Advanced Query Enhancement

In addition to the basic query enhancement features, Fusion has some advanced query enhancement features that are best configured by data scientists or the Lucidworks Professional Services team:

- *Smart Answers* analyzes your question-and-answer content to produce data that can power virtual assistants or chatbots.
- Fusion includes pre-trained *Sentiment Analysis and Prediction* models to assist with the interpretation of queries or content to determine some of their subjective aspects, such as emotions, intentions, opinions, and so on, using machine learning.

Smart Answers

Fusion Smart Answers brings the benefits of a versatile, scalable Semantic Search platform to provide cutting edge relevancy for your applications. This system makes use of advanced Deep Learning techniques to provide the power of Neural dense vectors search. Semantically rich models encode queries and documents into vectors in the same digital vector space in such way that query and the most relevant information are located near each other. It pushes search boundaries beyond classical token-based matching mechanisms by leveraging the contextual information and query understanding. Virtual assistants and Chatbots can incorporate Smart Answers to enable self-service for employees and customers and save cost on answering incoming queries.

E-commerce can utilize Smart Answers to handle zero-result search queries and improve the overall relevancy by recommending products that are most likely of users interest.

Training data can be explicitly provided as query/response pairs or constructed from the collected signals data. Even if you do not have an existing set of recorded interactions, you can rely on Smart Answers cold start solution. It uses various pre-trained models to get out-of-the-box (OOTB) semantic capabilities as well as provides a possibility to train unsupervised models for specific domains.

These features bring traditional search relevancy development and Data Science together into an easy-to-use configuration framework that leverages Fusion's indexing and querying pipelines.

Example business use cases

Call center or IT support
- Embed this system as a self-help feature on your Help page or Contact Us page to reduce the call center load.
- Make it available to your customer support team to find the answers to already-solved problems.

Zero-result search problem in E-commerce:
Queries that lead to zero-results is a huge problem for E-commerce that leads to income loss and decrease of the overall users experience. Semantic search does not have this problem as it does not operate on the exact token matches as classical search. Instead search is done in the vector space which can find relevant products even if there is no token overlapping.

Questions about products for E-commerce:
E-commerce websites can use this system to search "how to" content, product user manuals, or existing product questions. For example, amazon.com provides a search function on questions about each product.

Search in Slack, email conversations, or SharePoint FAQ docs
You can achieve fast knowledge extraction by applying this solution to these types of knowledge repositories.

Improve search for long queries
As the solution utilizes state of the art Deep Learning techniques, it is able to capture semantic and contextual information into query understanding. This makes it very suitable to work with long queries or natural language questions.

Getting started

To get started, you need a trained machine learning model. There are two methods for building a model, depending on the kind of data you already have:

The Supervised solution	The Cold Start solution
Use this solution when you already have a collection of query/response pairs or if you can construct such a dataset from the signals data. • Input: A dataset of query/response pairs. • Workflow: a. *Train a machine learning model* using the *Smart Answers Supervised Training* job. b. *Configure the pipelines*. c. *Evaluate the query pipeline*. d. *Check the job status and output*.	Use this solution when you have no historical training data or fewer than 200 query/response pairs. • Workflow 1: a. *Set Up A Pre-Trained Cold Start Model for Smart Answers* using the *Create Seldon Core Model Deployment* job. b. *Configure the pipelines*. c. *Evaluate the query pipeline*. d. *Check the job status and output*. • Workflow 2: a. Input: A body of content that can be used for unsupervised training. b. *Train a machine learning model* using the *Smart Answers Coldstart Training* job. c. *Configure the pipelines*. d. *Evaluate the query pipeline*. e. *Check the job status and output*.

Each method requires a slightly different model training procedure, but the model deployment procedure is the same for both.

Smart Answers workflow

To implement Smart Answers, you will follow this workflow:

1. Train or install a machine learning model.
   This process differs depending on whether you use the *Supervised Solution* or the *Cold Start Solution*.
2. Configure the index and query pipelines.
   Fusion includes default pipelines to get you started. See *Configure the Smart Answers Pipelines*.
3. Evaluate the query pipeline's effectiveness.
   Fine tune your query pipeline configuration by running a job that analyzes its effectiveness. See *Evaluate a Smart Answers Query Pipeline*.

Smart Answers jobs

These jobs provide the machine learning features that drive Smart Answers:

- Machine learning model training:
  - *Smart Answers Supervised Training* job. Fusion includes default pipelines to get you started. See *Configure the Smart Answers Pipelines*.
  - *Smart Answers Coldstart Training* job.
- Machine learning model deployment:
  - *Create Seldon Core Model Deployment*
  - *Delete Seldon Core Model Deployment*
- *Smart Answers Evaluate Pipeline*
  This job analyzes your configured Smart Answers query pipeline to provide insights about its effectiveness so that you can fine-tune your configuration for the best possible results.

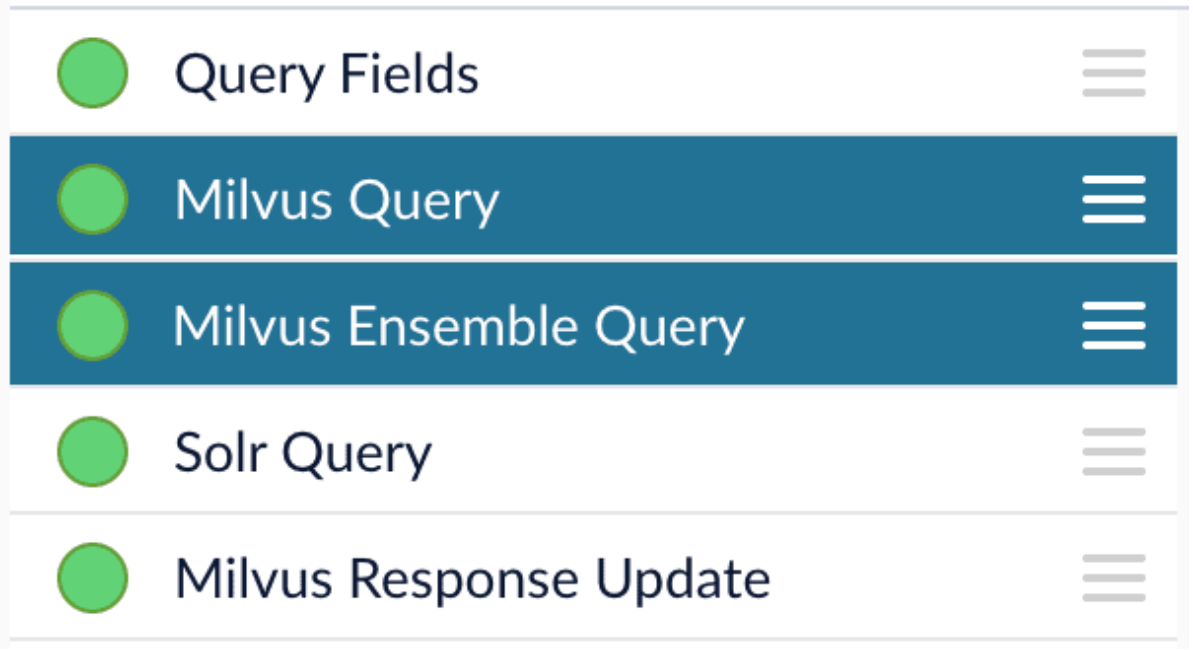
"Smart-answers" pipelines and stages

Once you have trained and deployed your model, you can use one of the default pipelines that are automatically created with your Fusion app. Both pipelines are called `APP_NAME-smart-answers`.

See *Configure the Smart Answers Pipeline* for more information.

- **"-smart-answers" index pipeline**

- **"-smart-answers" query pipeline**



Short answer extraction

By default, the question-answering query pipelines return complete documents that answer questions. Optionally, you can extract just a paragraph, a sentence, or a few words that answer the question. See *Extract Short Answers from Longer Documents*.

Best Practices

Below is a collection of useful tips and tricks that can help improve results.

How to begin

Use cases for chatbots, virtual assistants, QnA, and FAQ search

In most cases, Lucidworks recommends you begin with *coldstart models*.

- The coldstart model is only suitable for English. And the large coldstart model increases query time. So if a moderate query time is acceptable, use a *coldstart model*.
- For multilingual support or fast query time, use a multilingual model.

Coldstart models

Pre-trained coldstart models are:

- Robust out-of-the-box models.
- Easily configured and do not require training data.
- Used to establish a solid baseline solution to compare when running supervised training.

Use cases for never null and eCommerce

For these use cases, the preferred method is to build a training set and use a supervised job. It is usually possible to set up a training dataset. However, if that is not possible, set up an initial *coldstart pre-trained model*.

The quality of model and results may vary because models are designed to use natural language questions and not short, less defined queries. To enhance the results:

1. Collect signal data (if it does not exist).
2. Build a training set.

Obtain training data

There are multiple ways to obtain training data. The supervised training job uses pairs of questions and answers (or queries and product names) to train deep learning encoders.

Use cases for chatbots, virtual assistants, QnA, and FAQ search

For these use cases:

- Base training data typically consists of existing FAQ pair information.
- After the coldstart solution is deployed, training data can also be derived from sources such as search logs and customer support correspondence. The data can be constructed and expanded manually.
- Expanded training data can be obtained using the augmentation techniques in the *Data Augmentation job*. For natural language scenarios, backtranslation is effective to provide paraphrased versions of the original questions and answers that improve vocabulary and semantic coverage.

Use cases for never null and eCommerce

Training data for these use cases is primarily collected signals data.

There are different ways to construct training data, which determines how the resulting trained model behaves based on the:

- Available types of signals.
- Quantity of signals.
- Needs of your business.

Examples include:

1. Query and product pair click signals:
   - Help increase the click-through rate (CTR).
   - Are usually high volume, but low quality.
2. Add-to-cart and purchase signals:
   - Help increase sales.
   - Have a very strong relationship between query and products.

    - Are high quality, but low volume.
    - Produce a resulting trained model that ranks products most likely to be purchased higher than it does other products (in the results generated by the query).
3. Null-search signals result from products selected after a zero-result search or an abandoned search.
4. Different training datasets can be combined to train one model to rule them all or results from the different models can be combined at query time.

How to evaluate results

The supervised training job incorporates the evaluation mechanism. A hold out validation set:

- Is constructed automatically from unique questions and queries.
- Can be sized in the job configuration.
- Provides the ability to:
    - Evaluate how well models generalize to new, unseen queries.
    - Prevent the occurrence of overfitting.

You can also *Evaluate a Smart Answers Query Pipeline* to:

- Ensure configuration is set correctly.
- Compare different models or pipelines.
- Perform an end-to-end evaluation of the entire query pipeline.

How to choose a training model

Select the model that best fits your training needs.

RNN models

RNN models support most use cases, including digital commerce and heavy query traffic domains. The models also provide:

- Easily-understood training.
- Extremely fast query time.

While several model cases are available, Lucidworks recommends that in most cases, begin with `word_en_300d_2M`. If the case requires other language support, use `bpe_multilingual` or one of the specific `bpe` models for CJK languages.

BERT models

BERT models:

- Are *only* recommended if you have GPU access. Transformer-based models can *only* use a single GPU. Even if multiple GPUs are available, this type of model does *not* scale to multiple GPUs.
- Are suitable for natural language queries, chatbots, virtual assistants, and low query traffic domains.
- Provide better semantic quality results, especially if only a small amount of training data is available.
- Are more expensive to use for training.
- Run more slowly at query time than RNN models.

Model vector size

Pre-trained models produce vectors with a dimension of 512.

Vector dimensions for supervised job trained models are highlighted at the end of the training logs, or can be derived from the model and parameters:

- BERT-based model vectors have a dimension of 768.
- RNN-based model vectors have a dimension of two times the size of the last specified layer.

Hyperparameters to tune the training job

Most of the parameters in the supervised training job contain effective default values. Some of the parameters are automatically determined based on the dataset statistics.

Specific parameters to review are listed below, but see *Advanced Model Training Configuration for Smart Answers* for detailed information about tuning the training job.

General encoder parameters

These parameters are common to BERT and RNN models. The most significant parameter in this section is **Fine-tune Token Embeddings**. The parameter is disabled by default to prevent overfitting. However, enable the parameter to improve queries with multiple variations or jargon. For example, queries in the eCommerce domain.

RNN encoder parameters

Dropout Ratio
: Tune in the [0.1, 0.5] range to improve regularization and prevent overfitting. The customary value is 0.15 or 0.3.

RNN Function (Units) List
: The parameters determine the number of layers to use and the output vector size.

One layer is typically sufficient, and Lucidworks does *not* recommend using over two layers. Adding a smaller second layer produces a smaller vector size that reduces the vector index size and might reduce query time.

For example, the current configuration is a one-layer network with 128 units (256 vector size) that provides good results. To improve QPS and index a very large collection without losing quality, add a small layer with 64 units (128 vector size). Instead of adding one small layer, you can add two small layers. The two additional small layers slightly increases encoding time, but the vector size is two times smaller in memory, which may improve search time.

Training and evaluation parameters

To reduce training time, especially when training on a CPU, lower the value in following parameters:

- **Number of Epochs**
- **Monitor Patience**

To increase training time, increase the value in the parameters.

The **Training Batch Size** parameter impacts the timeframe of an epoch and how often validation is performed. Lucidworks recommends setting the following:

- At least 100 batches per training epoch
- Use bigger batches with larger training datasets
- Values of:
  - [32, 64] for small training sets
  - [128, 256] for medium training sets
  - [512, 1024, 2048] for large training sets

Due to the size of BERT-based models, the training batch size values recommended are [8, 16, 32].

How to choose a Milvus index

Milvus supports multiple indexes. See the following Milvus documentation:

- Vector index *(https://milvus.io/docs/v1.0.0/index.md#Vector-index)* for detailed descriptions of the indexes.
- Performance FAQ *(https://milvus.io/docs/v1.0.0/performance_faq.md)* for performance information.

Default index

The default (FLAT) index provides:

- The best possible quality (typically 100% of the model quality), but is the least effective in terms of query time.
- A reasonable query time for low traffic use cases under 250 QPS.

HNSW index

The HNSW index:

- Supports higher QPS.
- Supports collections over a few million documents.
- Needs to be configured with the parameter values detailed in *Create Indexes in Milvus Jobs*.

Cold Start Solution

When no training data exists to use the Supervised job, the cold start solution uses general pre-trained Deep Learning models or provides a possibility to train custom word embeddings for specific domains.

Over time, we suggest capturing *signals* from document clicks, likes, and downloads. These signals can be used to construct training dataset. After cumulating at least 3000 pairs, that feedback can be used as training data for the *Supervised solution*.

Like the Supervised solution, the Cold start solution has two parts:

1. Prepare a model

There are two ways to do this:

- Smart Answers comes with two pre-trained cold-start models. These models can serve as a strong baseline if your data does not have too much domain-specific terms. See *Set Up A Pre-Trained Cold Start Model*.
- You can train your own cold start model by configuring a job that analyzes your existing content. See *Train A Smart Answers Cold Start Model*.

2. Create collections in Milvus

In order to use `{app_name}-smart-answers` pipelines, you need to create collections in Milvus. Please refer to the *Milvus* documentation page.

3. Configure the pipelines

The trained model should be used at both index and query time in order to perform dense vector search.

- At index time, we provide the `{app_name}-smart-answers` index pipeline to help generate a dense vector representation of answers.
- At query time, we provide a `{app_name}-smart-answers` query pipeline to conduct run-time neural search. This pipeline transforms the incoming query into a dense vector using the trained model, then compares it with indexed answer dense vectors by computing the cosine distance between them. You can also use a query stage to combine Solr and document vector similarity scores at query time.

See *Configure The Smart Answers Pipelines*. Once your pipelines are configured, see *Evaluate a Smart Answers Query Pipeline* to test its effectiveness so that you can fine-tune the configuration.

FAQ Solution

The Supervised solution consists of two parts.

1. Train a model

Train a deep learning model using existing query/response pairs by configuring the Smart Answers Supervised Training job. See *Train A Smart Answers Supervised Model*.

The job includes an auto-tuning feature that seeks to identify the optimal configuration for your RNN-based model. Transformer-based models like BERT are not used in the auto-tuning.

2. Create collections in Milvus

In order to use `{app_name}-smart-answers` pipelines, you need to create collections in Milvus. Please, refer to the *Milvus* documentation page.

3. Configure the pipelines

The trained model should be used at both index and query time in order to perform dense vector search.

- At index time, we provide the `{app_name}-smart-answers` index pipeline to help generate a dense vector representation of answers.
- At query time, we provide a `{app_name}-smart-answers` query pipeline to conduct run-time neural search. This pipeline transforms the incoming query into a dense vector using the trained model, then compares it with indexed answer dense vectors by computing the cosine distance between them. You can also use a query stage to combine Solr and document vector similarity scores at query time.

See *Configure The Smart Answers Pipelines*. Once your pipelines are configured, see *Evaluate a Smart Answers Query Pipeline* to test its effectiveness so that you can fine-tune the configuration.

Advanced Model Training Configuration for Smart Answers

This topic provides tips for training your *Smart Answers* deep learning model.

Model Base

There are several types of model bases that can be used for training and fine-tuning:

- **word_en_300d_2M** are general pre-trained word embeddings. It is a good default choice to start with for English language.
- **bpe_{language}_{dim_size}_{vocab_size}** are general pre-trained BPE *(https://nlp.h-its.org/bpemb)* embeddings that are available for different languages, including CJK languages and multilingual. Also useful in scenarios when vocabulary is very big or when the data might have a lot of misspellings.
- **word_custom** or **bpe_custom** specifies that custom embeddings should be trained on users data via Word2Vec algorithm. It might be useful when your domain has a very unusual specific vocabulary.
- **transformer** based models such as **distilbert_{language}** and **biobert**. Much bigger and expensive models that might provide even better quality for FAQ, Chatbot and virtual Assistance use-cases. Also useful when the training data is limited.

If **word** or **bpe** based models are used, one or more RNN layers are added on top of the embeddings to be trained to capture contextual and semantic information. It is configurable in the **RNN Encoder Parameters** section. If you wish to use embeddings initialized on your data, refer to the **Custom Embeddings Initialization** to configure Word2Vec algorithm.

Transformer-based models already have specified fixed model architecture which is fine-tuned during the training procedure.

	Dimension size of vectors for Transformer-based models is 768. For RNN-based models it is 2 times the number units of the last layer. To find the dimension size: download the model, expand the zip, open the log and search for `Encoder output dim size:` line. You might need this information when creating collections in Milvus.
	We recommend to use Transformer-based models only if you can allocate GPU for the training job as these models are very computationally expensive.

At the end, Attention mechanism to aggregate output into final single vector for all models.

Auto hyperparameter tuning

By default, training module tries to select the most optimal parameter values (for those left as blank) based on the training data statistics. Auto-tune can extend it by automatically finding even better training configuration through hyper-parameter search.

If **Perform auto hyperparameter tuning** is enabled, multiple models will be trained across several stages. On each stage the most impactful parameters are tuned to find the best configuration. All other parameters are used with default values or those specified on UI.

Although this is a resource-intensive operation, it can be useful to identify better RNN-based configuration. Transformer-based models are not used during auto hyperparameter tuning as they have a fixed architecture. They usually perform better on Q&A tasks yet they are much more expensive on both training and inference time.

Input/Output parameters

Here you can specify the input data that should be used for training with possibility to filter or sample it.

	You can also configure this job to read from or write to cloud storage. See *Configure An Argo-Based Job to Access GCS* and *Configure An Argo-Based Job to Access S3*.

If you have additional text data that can be used for **custom embeddings initialization** or to learn and capture bigger vocabulary when **word_en_300d_2M** is used, please provide it in the **Texts Data Path** field.

Model Replicas parameter allows to specify how many replicas of the model should be deployed. Auto-balancing mechanism is used to distribute queries between model replicas, so more replicas might provide faster indexing as well as higher QPS.

If you use aggregated signals data for training or have weights for each training pair, you can also specify **Weight Field**. It will be used for sampling positive answers for a particular query if there are more than one possible. It is useful for eCommerce use-cases when for one unique query there might be a lot of different paired products.

	Weight Field will not be used if **Use Labelling Resolution** is set on. These parameters are mutually exclusive.

Data pre-processing parameters

Labeling Resolution allows to find missing query/response pairs in the training data which helps in the training. When set on, a graph of all pairs connections is built. Then connected components are obtained to match missing query/response pairs. For example if there are three existing pairs: `q1–a1`, `q2–a1` and `q2–a2`. Then Labelling Resolution will match `q1–a2` as additional pair through `q2–a1` connection. This is useful in Q&A use-cases when there are not a lot of answers per unique question, otherwise too big connected components will be found. If you have data when for one query there might be a lot of different responses, like in eCommerce, it is better to leave it off.

If **Use Labelling Resolution** is set on, **Weight Field** is ignored. These parameters are mutually exclusive.

The **Maximum vocabulary size**, **Lower case all words** and **Apply unidecode decoding** parameters impact the vocabulary size if `word_en_300d_2M`, `word_custom` or `bpe_custom` model bases are used. Otherwise these parameters are ignored and model specific pre-processing is used. Default values should work in most cases, given enough RAM and time to train.

If you want to train custom embeddings for languages like CJK, disable **Apply unidecode decoding**.

If you see an out-of-memory error, try reducing the vocabulary size and/or the training batch size. The **Minimum number of words** and **Maximum number of words** parameters can help trim problematic documents.

Custom embeddings initialization parameters

If **word_custom** or **bpe_custom** model bases are chosen, then custom embeddings will be trained on the provided data.

If you want to use addition dataset to train custom embeddings, please specify **Texts Data Path** and **Text Fields** in the **Input/Output parameters**.

Additionally, commonly-used Word2vec training parameters are **Word2Vec Training Epochs**, **Size of Word Vectors** and **Word2Vec Window Size**. Default values should work in most cases.

Smaller word vectors size makes models smaller and more robust to overfitting. However, dimensions smaller than 100 may impact the quality.

Evaluation parameters

Validation Sample Size controls how much *unique* queries should be hold-out and used for validation. It is a fraction if the value below 1.0 or specific number of queries if it is integer value higher than 1.

During evaluation, all responses/answers are used. They form an index which is queried by unique validation queries. **Eval ANN Index** parameter controls should it be ANN index or brute-force search with auto value by default. If you notice that evaluation takes a lot of time, try to enable ANN index or reduce the number of evaluation queries.

Generally, this evaluation setup is similar to how it will work in index and query pipelines, so the evaluation results should provide good approximation of the quality. To evaluate the configured pipelines on the test data, please use *Evaluate a Smart Answers Query Pipeline job*.

A list of evaluation metrics is provided to monitor the training process and measure the quality of the final model:

- Mean Average Precision (MAP)
- Mean Reciprocal Rank (MRR)
- Recall

You can choose from the list in the **Metrics list** parameter. It uses all metrics by default.

You can also specify measuring the ranking position for each metric. For example, if you specify **Metrics@k list** as `[1,3]`, with **Metrics list** `["map","mrr","recall"]`, then the metrics `map@1`, `map@3`, `mrr@1`, `mrr@3`, `recall@1`, and `recall@3` will be logged for each training epoch and final model.

You can choose a particular metric at a particular `k` (controlled by the **Monitoring metric** parameter) to help decide when to stop training. Specifically, when there is no increase in the Monitoring metric value for a particular number of epochs (controlled by the **Patience during monitoring** parameter), then training stops.

During the training we evaluate the result using similar cold-start model (weighted average of word vectors) as a baseline. Look for the *Cold-start encoder validation evaluation* section of the logs, it is printed before first training epoch.

General Encoder parameters

Note that the following parameters are common across all model bases including RNN and Transformer architectures.

- **Fine-tune Token Embeddings** will allow to fine-tune embeddings (word vectors) layer to be updated during the training alongside with all other layers. It is disabled by default as it is usually one of the biggest layer in the network and updating it might lead to overfitting. It is useful to enable if your data have a lot of specific or misspelled words.
- **Max Length** controls the maximum context window that model can process. Texts longer than this value will be trimmed. The default value is the max value between three times the STD of question lengths and two times the STD of answer lengths.
  The longer the context the longer and harder it takes for model to process. This parameter is especially important for Transformer-based models as it affects training and inference time. Note that the maximum supported length for Transformer models is 512 tokens, so you can specify any value up to that.
- **Global Pool Type** specifies how token vectors should be aggregated to obtain final content vector. The default mechanism is self-attention which provides the best quality in most cases.

- **Number of clusters** and **Top K of clusters to return** are deprecated since 5.3 and will be removed in the following releases. There is no practical need to use them after Milvus vectors similarity search integration.

RNN Encoder parameters

We use RNN-based deep learning *(https://en.wikipedia.org/wiki/Recurrent_neural_network)* architecture for `word` and `bpe` model bases, with the flexibility to choose between `LSTM` and `GRU` layers with more than one layer. We don't recommend using more than three layers. The layers and layer sizes are controlled by the **RNN function list** and **RNN function units list** parameters.

Dropout ratio parameters provides regularization effect and is applied between embeddings layer and the first RNN layer.

Training parameters

These parameters controls the training procedure. Most of them are left blank so the robust default values can be determined by the training module based on the dataset statistics.

The learning rate scheduler has 3 stages. Firstly, it linearly increases the LR from **Minimum Learning Rate** value to **Base Learning Rate** value over **Number of Warm-Up epochs**. Then it stays consistent for **Number of Flat epochs**. And at the last stage Cosine Annealing is used for the remain number of epochs.

Use Mixed Precision parameter can enable mixed precision during the training for Transformer-based models if modern GPU are used (Turing and later). It helps to get more VRAM so bigger batch size can be used. As well as provides some performance boost in training time.

Cross-Batch Memory parameters allow to re-use encoded representations from the previous batches during loss computation, so loss function can process more positive and negative examples for the model update. It works well with Transformer-based models that consumes more VRAM and can be used with only with limited batch size. This is not necessary when the training batch size is large. When configured, this number needs to be greater than or equal to the training batch size.

Sentiment Analysis and Prediction

One major part of understanding intent is understanding sentiment. Fusion has done the heavy lifting by integrating state-of-the-art, pre-trained sentiment analysis models directly into the platform. Lucidworks makes these models readily available for search developers to embed sentiment scoring into the query and indexing pipelines as stages.

What are sentiment analysis and sentiment prediction?

Sentiment analysis is the interpretation of queries or content to determine some of their subjective aspects, such as emotions, intentions, opinions, and so on, using machine learning. Sentiment analysis produces a sentiment model that Fusion can use to perform sentiment prediction.

Since stronger sentiments can be positive or negative while a weaker sentiment is more neutral, sentiment analysis uses the positive/negative polarity to measure a sentiment's strength:

```
Strongly Strongly Negative Neutral Positive −2 −1 0 1 2
```

For example, a query on a customer support site can express negative emotions, as in "How can I fix this frustrating upgrade problem?" Similarly, a shopper may imply a positive or negative intention to purchase an item, or a positive or negative opinion of that item.

Intelligent applications with sentiment analysis and prediction

When you assign sentiment values to incoming queries or to documents at indexing time, you can leverage that data to deliver more insight to your business applications. Understanding the intent and sentiment of customers or employees can help determine which information you deliver or how you treat subsequent interactions in the workflow. This kind of intelligence is valuable for e-commerce, call centers, social media, or surveys.

Pre-trained sentiment models

Lucidworks provides two pre-trained sentiment models that you can deploy to get started:

sentiment-general:v1.0	**sentiment-reviews:v1.0**
A general-purpose model, trained on short sentences. Suitable for short texts and for intent prediction. See *Deploy The sentiment-general:v1.0 Model* for instructions.	A model trained on a variety of customer reviews. Optimized for longer texts. It also supports highlighting the tokens that provide stronger sentiment. See *Deploy The sentiment-reviews:v1.0 Model* for instructions.

These models support the English language only.

Model output

Both of the pre-trained models output the following:

- a **label**, `negative` or `positive`
- a **score** from `−2` to `2`

The sentiment-reviews:v1.0 model also optionally outputs tokens and their corresponding attention weights, that is, the weight that each token carries in the sentiment prediction. The total of all attention weights is always 1, that is, each value represents a percentage of the total weight. In the example below, "awesome" has the highest weight because it expresses the strongest sentiment compared to other tokens in the string:

```
// Input text = "That is awesome!" // Output sentiment_label = ‘positive’
sentiment_score = 1.998 sentiment_attention_tokens = ['That', "'", 's', 'awesome',
'!'] sentiment_attention_weights = [0.154, 0.078, 0.069, 0.444, 0.255]
```

Operations

To ensure Fusion operates efficiently and successfully, use the information in these topics to install and configure Fusion correctly and monitor system processes.

Fusion 5 Survival Guide

The *Fusion 5 Survival Guide* details how to install, configure, and run Fusion 5 in production in high availablity on Kubernetes. This comprehensive cloud operations guide is curated by the Fusion Engineering Team.

Security options available

Security is essential to ensure successful operations.

The *Security concept page* provides general information about authentication and authorization of users, as well as links to more detailed instructions.

Monitoring functions

After Fusion is configured and operational, monitoring is required to maintain successful platform operations. The *Monitoring* section provides information and links to observe, investigate, and troubleshoot issues.

Fusion jobs and job scheduling

Jobs are system tasks that load and aggregate data, train models, and perform other activities to help keep your search platform running efficiently. For more information, see *Jobs and scheduling*.

Fusion 5 Survival Guide

The purpose of this guide is to help you install, configure, and run Fusion 5 in production with high availability on Kubernetes. You can also find this guide in the fusion-cloud-native repo *(https://github.com/lucidworks/fusion-cloud-native/)*.

	The Fusion 5 Survival Guide prefers Google Kubernetes Engine (GKE) terminology. Although your Kubernetes provider's terminology may differ, the general concepts are consistent between providers.

Foundational concepts

- *Which Kubernetes?*
- *Multiple Zones for High Availability*
- *Fusion Microservices*
- *Ingress and Security*
- *Stateless Sessions with JWT*
- *Workload Isolation with Multiple Node Pools*
- *High-Performance Query Processing with Auto-Scaling*
- *Fusion 5 Frequently Asked Questions*

Planning your deployment

- *Deploy Fusion at Scale*

"Day two" operations

- *Integrate Fusion with Your Logging Infrastructure*
- *Configure Pod Affinity*
- *Configure Resource Limits*
- *Spark Operations*
- *Fusion Upgrades*

Which Kubernetes?

> The image above is for illustrative purposes only. It depicts services available in Fusion 5.11, but your actual implementation may differ.

To help users get started with Fusion 5, Lucidworks provides setup scripts for Google Kubernetes Engine (GKE), Amazon Elastic Kubernetes Service (EKS), and Azure Kubernetes Service (AKS). It's possible to use other Kubernetes providers, but the instructions and details provided in this article target the Kubernetes providers listed within release notes. To run Fusion 5, use your cloud provider's stable release channel.

> Although GKE refers to the latest stable version as the "stable release channel," your cloud provider may use different terminology.

For more information, see:

- GKE Release Channels *(https://cloud.google.com/kubernetes-engine/docs/concepts/release-channels)*
- EKS Kubernetes Release Calendar *(https://docs.aws.amazon.com/eks/latest/userguide/kubernetes-versions.html#kubernetes-release-calendar)*
- AKS Kubernetes Release Calendar *(https://docs.microsoft.com/en-us/azure/aks/supported-kubernetes-versions#aks-kubernetes-release-calendar)*

More information

Here are instructions for deploying on some popular Kubernetes platforms:

- *Google Kubernetes Engine (GKE)*
- *Amazon Elastic Kubernetes Service (EKS)*
- *Azure Kubernetes Service (AKS)*
- *Other Kubernetes Platforms*

Multiple Zones for High Availability

Fusion relies on ZooKeeper to keep quorum. As a result, you need at least three ZooKeeper pods in an ensemble. ZooKeeper deploys as a StatefulSet in the Fusion Helm chart. With GKE, you can launch a regional cluster *(https://cloud.google.com/kubernetes-engine/docs/concepts/regional-clusters)* that distributes nodes across three availability zones. With a multi-zone setup, your cluster can withstand the loss of one zone without experiencing downtime. However, you may experience degraded performance from losing one third of your total compute capacity, assuming your pods are evenly distributed. The multi-zone cluster also ensures higher availability for the Kubernetes control plane services.

Lucidworks provides affinity rules to ensure multiple pods per service get distributed across multiple zones. See *Configure Pod Affinity* for details on configuring.

When running in a multi-zone cluster, each Solr node has a `solr_zone` system property set to the zone it is running in, such as `-Dsolr_zone=us-west1-a`. This guide covers how to use the `solr_zone` property to distribute replicas across zones in the *Solr Auto-scaling Policy* section. Setting the `solr_zone` property for Solr pods requires the Solr service account to have a ClusterRoleBinding that allows it to get node metadata from the Kubernetes API service.

Ingress and Security

The Fusion proxy service serves as an API gateway and provides authentication and authorization. Route all external access to Fusion services through the Fusion proxy service. The most common approach is to set up a Kubernetes Ingress *(https://kubernetes.io/docs/concepts/services-networking/ingress/)* that routes Fusion service requests to the proxy service as shown in the example ingress definition *below*. It's also common to do TLS termination *(https://cloud.google.com/load-balancing/docs/https/#tls_support)* at the Ingress, so that all traffic to and from the Kubernetes cluster is encrypted.

Internal requests happen over unencrypted HTTP.

```
apiVersion: v1 items: - apiVersion: extensions/v1beta1 kind: Ingress metadata:
annotations: ... labels: ... name: <RELEASE>-api-gateway namespace: <NAMESPACE>
spec: rules: - host: <HOSTNAME> http: paths: - backend: serviceName: proxy
servicePort: 6764 path: "/*" tls: - hosts: - <HOSTNAME> secretName: <RELEASE>-api-
gateway-tls-secret status: loadBalancer: ingress: - ip: <SOME_IP>
```

EKS

For deployments that run on EKS, the setup_f5_eks.sh *(https://github.com/lucidworks/fusion-cloud-native/blob/master/setup_f5_eks.sh)* setup script creates a classic elastic load balancer (ELB) to expose the Fusion proxy service for demonstration purposes. However, you're strongly encouraged to configure a Kubernetes Ingress with TLS termination in front of the proxy service. See How do I terminate HTTPS traffic on Amazon EKS workloads with ACM? *(https://aws.amazon.com/premiumsupport/knowledge-center/terminate-https-traffic-eks-acm/)* for more information.

You can also opt to use an Application Load balancer *(https://github.com/lucidworks/fusion-cloud-native#eks-ingress)* instead of the proxy service.

AKS

AKS deployments can use the setup_f5_aks.sh *(https://github.com/lucidworks/fusion-cloud-native/blob/master/setup_f5_aks.sh)* setup script, which exposes the Fusion proxy service on an external IP using HTTP for demonstration purposes. We recommend that you configure a Kubernetes Ingress with TLS termination *(https://github.com/lucidworks/fusion-cloud-native#aks-ingress)* in front of the proxy service.

Other Kubernetes Platforms

If your deployment runs on OpenShift using a cluster created with OpenShift Infrastructure Provider, the setup_f5_k8s.sh *(https://github.com/lucidworks/fusion-cloud-native/blob/master/setup_f5_k8s.sh)* setup script will install Fusion 5 on your existing cluster. OpenShift Online services are not supported.

See the RedHat OpenShift *(https://github.com/lucidworks/fusion-cloud-native#redhat-openshift)* section of the readme for more tips toward a successful installation.

If your deployment runs on GKE, the setup_f5_gke.sh *(https://github.com/lucidworks/fusion-cloud-native/blob/master/setup_f5_gke.sh)* setup script in the `fusion-cloud-native` repo provides the option to create the Ingress and TLS cert *(https://github.com/lucidworks/fusion-cloud-native#gke-ingress-and-tls)*.

This process uses Let's Encrypt *(https://letsencrypt.org/)*.

For other Kubernetes providers, refer to provider's documentation on creating an Ingress and TLS certificates.

Stateless Sessions with JWT

The Fusion API gateway requires incoming requests to be authenticated. The gateway supports a variety of authentication mechanisms, including Security Assertion Markup Language (SAML), OpenID Connect (OIDC), Kerberos, and Basic authentication. Once authenticated, the gateway issues a JWT and returns it in the `id` cookie.

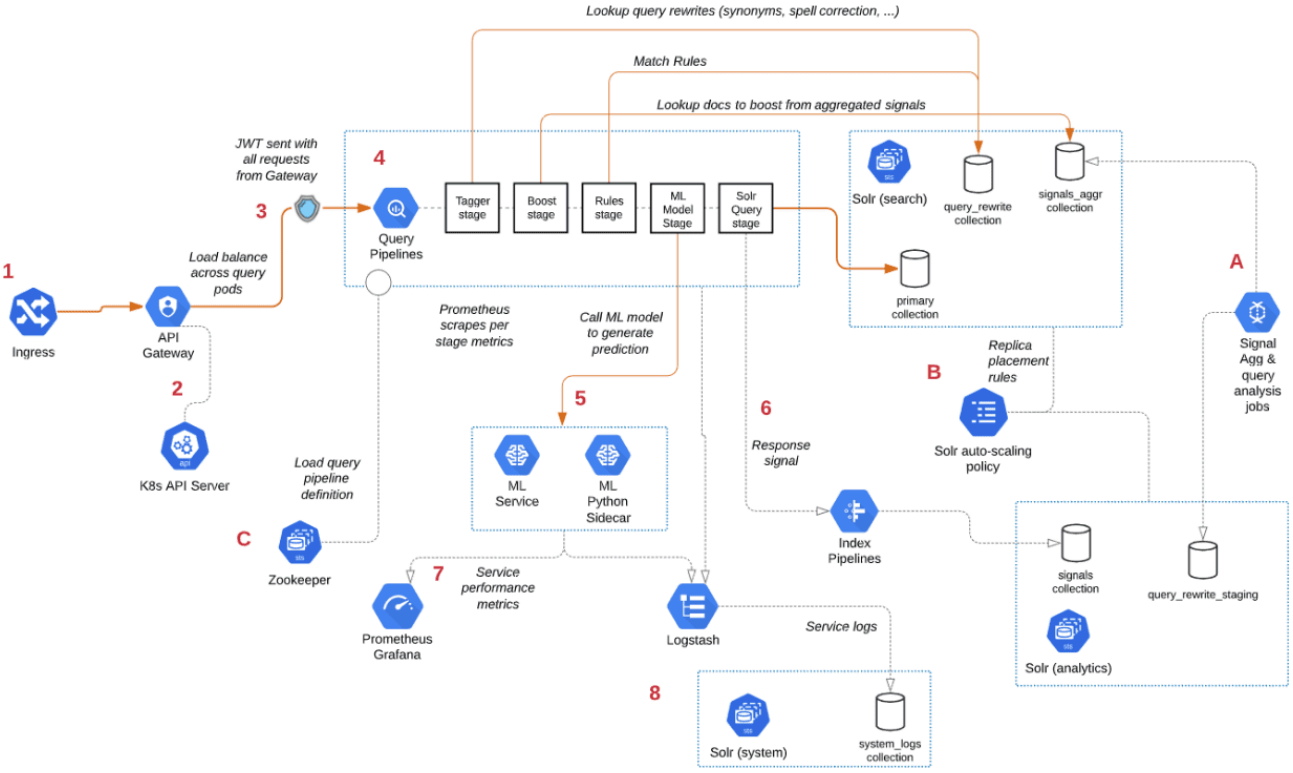
Client applications get the best performance by using the `id` cookie (or JWT Authorization header) instead of using Basic authentication for every query request. Verifying a JWT is fast and safe to cache. Hashing a password is CPU intensive and slow. (Fusion uses bcrypt *(https://en.wikipedia.org/wiki/Bcrypt)*.)

All Fusion services require requests to include a JWT to identify the caller.

High-Performance Query Processing with Auto-Scaling

To further illustrate key concepts about the Fusion 5 architecture, this section describes how query execution works and the microservices involved. Note some important details:

- There are many microservices involved in query execution. This illustrates the value and importance of having a robust orchestration layer like Kubernetes.
- Fusion comes pre-configured. You don't need to configure all details depicted in *Figure 2* below.



- **Point A** - Background Spark jobs aggregate signals—which powers the signal boosting stage—and analyzes signals for *query rewriting operations*. This includes head/tail, synonym detection, and misspelling detection.
- **Point B** - Fusion uses a Solr auto-scaling policy *(https://solr.apache.org/guide/solrcloud-autoscaling-overview.html)* in conjunction with Kubernetes node pools to govern replica placement for various Fusion collections. For example, to support high performance query traffic, place the primary collection with sidecar collections for query rewriting, signal boosting, and rules matching. Solr pods supporting high volume, low-latency reads are backed by an HPA linked to CPU or custom metrics in Prometheus. Fusion services store configurations, such as query pipeline definitions, in ZooKeeper (**Point C**).
- **Point 1** - A query request comes into the cluster from a Kubernetes Ingress *(https://cloud.google.com/kubernetes-engine/docs/concepts/ingress)*. The Ingress routes requests to the Fusion API Gateway service. The gateway performs authentication and authorization to ensure the user has the correct permissions to execute the query. The Fusion API Gateway load-balances requests across multiple query pipeline services using native Kubernetes service discovery (**Point 2**).
- **Point 3** - The gateway issues a JWT, which is sent to downstream services. An internal JWT has identifying information about a user, including their roles and permissions, to allow Fusion services to perform fine-grained authorization. The JWT is returned as a Set-Cookie header to improve performance of later requests. Alternatively, API requests can use the `/oauth2/token` endpoint in the gateway to retrieve the JWT using OAuth2 semantics.
- **Point 4** - The query service executes the pipeline stages to enrich the query, before sending it to the primary collection. Typically, this involves multiple lookups to sidecar collections—such as the `APP_NAME_query_rewrite` collection—to perform spell correction, synonym expansion, and rules matching. The query pipeline may also call out to the Fusion ML Model service to generate predictions, such as deciding query intent. The ML Model service may also use an HPA tied to CPU, to scale out and support the desired queries per second (**Point 5**).
- **Point 6** - After executing the query in the primary collection, Fusion generates a **response** signal to track query request parameters and Solr response metrics, such as `numFound` and `qTime`. Fusion stores raw signals in the **signals** collection, which typically runs in the analytics partition to support high-volume writes.
- **Point 7** - Every Fusion microservice exposes detailed metrics. Prometheus scrapes the metrics using pod annotations. The query microservice exposes stage-specific metrics to help you understand query performance.
- **Point 8** - Every Fusion microservice sends logs to Logstash. You can configure Logstash to index log messages into the `system_logs` collection in Solr or to an external service like Elastic.

Workload Isolation with Multiple Node Pools

You can run all Fusion services on a single Node Pool *(https://cloud.google.com/kubernetes-engine/docs/concepts/node-pools)*. Kubernetes does its best to balance resource use across the nodes. However, Lucidworks recommends defining multiple node pools to separate services into "workload partitions" based on the type of traffic a service receives.

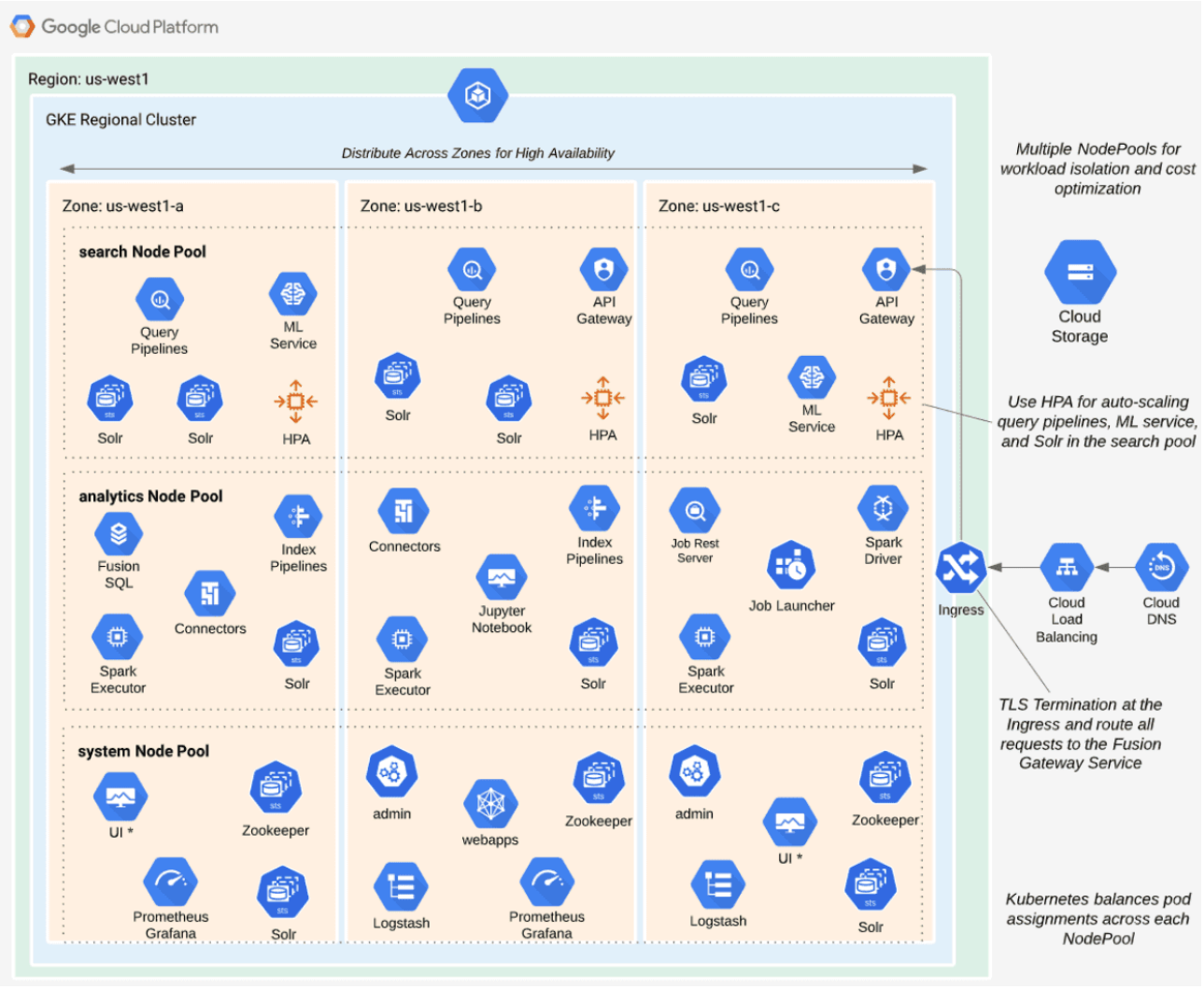
The Fusion Helm chart supports three optional partitions: ***search***, ***analytics***, and ***system***. Workload isolation with node pools allows you to optimize resource use across the cluster to achieve better scalability, balance, and minimize infrastructure costs. It also helps with monitoring, as you have better control over the traffic handled by each node in the cluster. To use this feature, define separate node pools in your Kubernetes cluster.

	The **analytics**, **search**, and **system** partitions are recommended starting points. You can extend upon this model to refine your pod allocation by adding more node pools. For example, running Spark jobs on a dedicated pool of preemptible nodes is a successful pattern in Kubernetes clusters at Lucidworks.

search

The following diagram depicts how the **search** partition hosts the API gateway (proxy), query pipelines, and machine learning (ML) model service. It also hosts a Solr StatefulSet that hosts collections to support high volume, low-latency reads. This includes your primary search collection and the *signals_aggr* collection, which serves signal boosting lookups during query execution.

When using multiple node pools to isolate or partition workloads, the Fusion Helm chart defines multiple StatefulSets for Solr. Each Solr StatefulSet uses the same ZooKeeper connect string, so they are considered to be in the same Solr cluster. A Solr autoscaling policy partitions the collections based on workload and zone. The autoscaling policy also ensures even distribution of replicas between multiple availability zones (typically three for HA). This ensures your Fusion cluster can withstand the loss of one availability zone and remain operational.



analytics

The **analytics** partition hosts the Spark driver and executor pods, Spark job management services (`job-rest-service` and `job-launcher`), and index pipelines. It also hosts a Solr StatefulSet for hosting analytics-oriented collections, such as the signals

collection. The signals collection typically experiences high write volume and batch-oriented read requests from Spark jobs that do large table scans on the collection throughout the day. In addition, the analytics Solr pods may have different resource requirements than the search Solr pods. You don't need as much memory for analytics pods, as they're not serving facet queries and other memory intensive workloads.

	When running in GKE, separating the Spark driver and executor pods into a dedicated node pool backed by preemptible nodes is a common pattern for reducing costs while increasing the compute capacity for running Spark jobs. You can also do this on EKS with spot instances. See *Spark Administration in Kubernetes* for more details.

system

The **system** partition hosts all other Fusion services, including the stateless UI services (`rules-ui`), Prometheus, and Grafana. It also hosts Solr pods that host system collections like `system_blobs`. Lucidworks recommends running your ZooKeeper ensemble in the system partition.

GPU Support for Deep Learning Training

In Fusion 5.3 and later, jobs that involve training deep learning-based models automatically use GPU resources for training if deployed on a GPU-enabled node. Your cloud provider likely has a custom set of nodeSelectors and tolerations required to map these jobs to their GPU compute. The following section provides an example using Smart Answers and GKE.

Training Smart Answers on GPU with GKE

To use GPU resources within GKE for Smart Answers training, create a GPU resource within your cluster.

Create a new nodepool with a pre-emptible GPU node that will spin down when not in use. Give the nodepool a label of `node_pool:gpu`. By default, GKE will also add a taint of `nvidia.com/gpu:present=NoSchedule`. Consider that fact when updating your Helm char values.

Additionally, add a specific resource limit of `nvidia.com/gpu: 1 `. (This value is specific to GKE.) Create another standard nodepool without GPU resources with a label of `node_pool:deploy` for the eventual Seldon Core deployment.

In your custom values YAML file, add:

```
question-answering: nodeSelector: default: cloud.google.com/gke-nodepool: gpu-
nodepool supervised: seldon: cloud.google.com/gke-nodepool: cpu-nodepool coldstart:
seldon: cloud.google.com/gke-nodepool: cpu-nodepool tolerations: default: - key:
"nvidia.com/gpu" operator: "Equal" value: "present" effect: "NoSchedule"
supervised: seldon: [] coldstart: seldon: [] resources: supervised: train:
requests: nvidia.com/gpu: 1 limits: nvidia.com/gpu: 1 coldstart: train: requests:
nvidia.com/gpu: 1 limits: nvidia.com/gpu: 1
```

> This setup deploys all workflow steps onto the GPU node *except* for the Seldon Core deployment. As the deployment will live on after the workflow has completed, assigning the Seldon Core deployment to the GPU node it would prevent GKE from spinning the GPU node down. This increases operating expense.

Setting up Milvus on GPU with GKE

Setting up Milvus on GPU first requires the creation of a GPU resource.

At the end of the `ml-model-service` section of your custom values YAML file, add a section for Milvus as shown below:

```
ml-model-service: # ml-model-service yaml settings: # ... # followed by the Milvus
settings: milvus: gpu: enabled: true image: repository: milvusdb/milvus tag:
0.10.2-gpu-d081520-8a2393 pullPolicy: "IfNotPresent" resources: requests:
nvidia.com/gpu: 1 limits: nvidia.com/gpu: 1 nodeSelector: cloud.google.com/gke-
nodepool: gpu-nodepool tolerations: - key: "nvidia.com/gpu" operator: "Equal"
value: "present" effect: "NoSchedule"
```

The example above assumes the name of the GPU nodepool is `gpu-nodepool`.

> The taints/tolerations and resource keys shown below are for a GKE setup. These values may vary depending on your cloud provider.

Fusion 5 On-prem Kubernetes Deployment

The `setup_f5_k8s.sh` script *(https://github.com/lucidworks/fusion-cloud-native/blob/master/setup_f5_k8s.sh)* in the `fusion-cloud-native` *(https://github.com/lucidworks/fusion-cloud-native)* repository provides deployment support for any Kubernetes platform, including on-premise, private cloud, public cloud, and hybrid platforms.

This script is used by the `setup_f5_gke.sh` *(https://github.com/lucidworks/fusion-cloud-native/blob/master/setup_f5_gke.sh)*, `setup_f5_eks.sh` *(https://github.com/lucidworks/fusion-cloud-native/blob/master/setup_f5_eks.sh)*, and `setup_f5_aks.sh` *(https://github.com/lucidworks/fusion-cloud-native/blob/master/setup_f5_aks.sh)* scripts, which provide additional platform-specific support for Google Kubernetes Engine (GKE), Amazon Elastic Kubernetes Service (EKS), and Azure Kubernetes Service (AKS).

See *Fusion 5 Upgrades* for more information about deployment instructions for:

- Natively supported deployments
- Other Kubernetes deployments

Considerations for on-prem deployments

Solr

Making changes to Solr's memory settings after the initial installation, including changing the number of pods, is simple. Changing the size and type of disk, however, can be difficult. Before you begin, identify some basic sizing information for Solr:

- Memory settings
- Disk size
- Disk type (e.g. SSD)

Docker images

Fusion's public Docker images are hosted on DockerHub. If you do not allow your Kubernetes environment to pull images from DockerHub, you will need to copy the images to your internal registry. This can be a time-consuming process.

Helm charts

Helm is a package manager for Kubernetes that helps you install and manage applications on your Kubernetes cluster. We supply a Helm chart for the Fusion installation.

If you are using Helm V2, the `setup_f5_k8s.sh` script installs Helm's Tiller component into your Kubernetes cluster with the cluster admin role. If you want to prevent the installation of Tiller, upgrade to Helm V3 *(https://v3.helm.sh/docs/topics/v2_v3_migration)*.

Nodes

We recommend installing Fusion on at least 3 nodes, each starting in its own namespace. Only 1 Fusion instance per namespace is supported.

Development and test environments only require 1 node with at least 12 GB of RAM.

It is possible to install other apps alongside Fusion in the same namespace, if desired.

Pod resources

We recommend not placing resource limits on pods until you have run production-like workloads with your data and fully understand your resource requirements.

Fusion gateway service

If you will be exposing the Fusion gateway service externally, you will need to consider your Kubernetes Ingress strategy:

- How will you route to internal services, such as the gateway?
- Will you do TLS termination at the Ingress?

Spark Administration in Kubernetes

In Fusion 5.x, Spark operates in native Kubernetes mode *(https://spark.apache.org/docs/latest/running-on-kubernetes.html)*, rather than the standalone mode used in Fusion 4.x. This topic describes Spark operations in Fusion 5.x.

Node Selectors

You can control which nodes Spark executors are scheduled on using a Spark configuration property for a job:

```
spark.kubernetes.node.selector.<LABEL>=<LABEL_VALUE>
```

Use the `LABEL` specified for the node, and the name of the node as the `LABEL_VALUE`. For example, if a node is labeled with `fusion_node_type=spark_only`, schedule Spark executor pods to run on that node using:

```
spark.kubernetes.node.selector.fusion_node_type=spark_only
```

	Spark version 2.4.x does not support tolerations for Spark pods. As a result, Spark pods can't be scheduled on any nodes with taints.

Cluster mode

Fusion 5 ships with Spark and operates in "cluster mode" on top of Kubernetes. In cluster mode, each Spark driver runs in a separate pod, and resources can be managed per job. Each executor also runs in its own pod.

Spark config defaults

The table below shows the default configurations for Spark. These settings are configured in the job-launcher config map, accessible using `kubectl get configmaps <release-name>-job-launcher`. Some of these settings are also configurable via Helm.

Spark Resource Configurations

Spark Configuration	Default value	Helm Variable
`spark.driver.memory`	3g	
`spark.executor.instances`	2	`executorInstances`
`spark.executor.memory`	3g	
`spark.executor.cores`	6	
`spark.kubernetes.executor.request.cores`	3	
`spark.sql.caseSensitive`	true	

Spark Kubernetes Configurations

Spark Configuration	Default value	Helm Variable
`spark.kubernetes.container.image.pullPolicy`	Always	`image.imagePullPolicy`
`spark.kubernetes.container.image.pullSecrets`		`image.imagePullSecrets`
`spark.kubernetes.authenticate.driver.serviceAccountName`	<name>-job-launcher-spark	
`spark.kubernetes.driver.container.image`	fusion-dev-docker.ci-artifactory.lucidworks.com	`image.repository`
`spark.kubernetes.executor.container.image`	fusion-dev-docker.ci-artifactory.lucidworks.com	`image.repository`

Spark operations how-tos

These topics provide how-tos for Spark operations:

- *Configure Spark Job Resource Allocation*
- *Configure Spark Credentials in The Kubernetes Cluster*
- *Get Logs for A Spark Job*
- *Clean Up Spark Driver Pods*
- *Install The Spark History Server*
- *Configure The Spark History Server*
- *Access The Spark History Server*

Kubernetes Deployment

The figure below depicts a Fusion 5 cluster running in a single Kubernetes namespace for high availability:

> The image above is for illustrative purposes only. It depicts services available in Fusion 5.11, but your actual implementation may differ.

Architecture and requirements

- *Which Kubernetes?*
- *Multiple Zones for High Availability*
- *Fusion Microservices*
- *Ingress and Security*
- *Stateless Sessions with JWT*
- *Workload Isolation with Multiple Node Pools*
- *High-Performance Query Processing with Auto-Scaling*

Deploy Fusion on Kubernetes

- *Deploy Fusion 5 in Google Kubernetes Engine*
- *Deploy Fusion 5 in Amazon Elastic Kubernetes Service*
- *Deploy Fusion 5 in Azure Kubernetes Service*
- *Deploy Fusion 5 on Other Kubernetes Platforms*
- *Deploy Fusion at Scale*

Upgrade Fusion on Kubernetes

Upgrade information about Kubernetes is included in the *Fusion 5 Upgrades* topic.

Jobs and Scheduling

A *job* is a runnable Fusion object that performs a specific activity. Datasource jobs, Spark jobs, and tasks are all types of jobs. Examples of the activities include:

- Aggregating signal data
- Training a shallow neural model and projecting each document onto the resulting vector embedding space
- Loading data into a data source

Job types

Each job type is a type of Fusion object that you can run or schedule to be run.

`datasource`	A job to ingest data according to the specified datasource configuration, such as `datasource:movie-db`. Datasources are created using the *Connector Datasources API* or the Fusion UI. See *Datasource Jobs*.
`spark`	A Spark job to process data, such as `spark:dailyMetricsRollup-counters`. Spark jobs are created using the *Spark Jobs API* or the Fusion UI. See *Spark Jobs*.
`task`	A job to perform an HTTP call or log cleanup, such as `task:delete-old-system-logs`. Tasks are created using the Fusion UI. See *Tasks*.

The Jobs manager

The Jobs manager, available in the Fusion workspace at **Collections > Jobs**, provides a simple interface for viewing and scheduling jobs. You can also create tasks and Spark jobs here.

	You cannot create, run, or schedule datasource jobs in the Jobs manager. You must create them on the Datasources page, in the Index Workbench, or in the Quickstart. To run a datasource job, use the *scheduler*.

Additional resources

- View our training on Jobs and Scheduling *(https://academy.lucidworks.com/jobs-and-scheduling)*.

Fusion Schedules

A *schedule* defines when Fusion will perform a job.

You can schedule a job using `cron` notation, ISO-8601 interval definitions, or triggers that depend on the results of other jobs.

You can also run jobs on demand.

The *Jobs API* lets you view jobs and their run histories, configure their schedules, and control them directly.

The Scheduler

The Scheduler, available in the Fusion workspace at **System > Scheduler**, provides an interface for scheduling jobs.

You cannot create jobs in the Scheduler. To create new jobs, use the *Jobs manager* or the REST API for each *job type*.

Datasource Jobs

A datasource is a configuration that manages the import and indexing of data into the collection.

Datasources are created using the *Index Workbench*, or at **Indexing > Datasources**, or using the *Connector Datasources API*.

A datasource is also a job that can run on demand or on a schedule.

Job History

Checking a job status

With the API

You can see the current job states of all datasources for the app `myApp` by calling the Jobs API:

```
curl -u USERNAME:PASSWORD http://proxy-url:6764/api/apps/myApp/jobs?
type=datasource
```

If a job has never run, a Jobs API call returns the following:

```
[{ "resource": "datasource:myDatasource", "enabled": true, "startedBy": "09765f1d-
2f1c-4c3b-ba22-887e3886d8aa", "status": "ready", "extra": {} }]
```

The value **`datasource:myDatasource`** indicates a datasource, **`myDatasource`**, is present. The line **`"status": "ready"`** indicates the current state of the job, which has never run.

After a job successfully completes, the following is returned:

```
[{ "resource": "datasource:myDatasource", "enabled": true, "startedBy": "09765f1d-
2f1c-4c3b-ba22-887e3886d8aa", "status": "success", "extra": { "counter.failed": 0,
"counter.other.pipeline.in": 58242, "counter.stage.field-mapping::d3ebd921-6065-
479e-9d45-0bfed819346b.processed": 58241, "counter.deleted": 0,
"counter.stage.solr-dynamic-field-name-mapping::e6aa9c51-2867-40a0-97fd-
34d909098a20.processed": 58241, "counter.new": 2268, "counter.output": 58241,
"counter.other.pipeline.complete": 58242, "datasourceId": "myDatasource",
"counter.input": 60435, "startTime": 1586298855843, "endTime": 1586299313433,
"counter.skipped": 2194, "counter.stage.solr-index::01e1c561-0440-44b2-afb4-
d6ff471f646c.processed": 58241, "counter.other.pipeline.out": 58242 },
"lastStartTime": "2020-04-07T22:34:15.852Z", "lastEndTime": "2020-04-
07T22:41:59.271Z" }]
```

To see the complete job history, use the following Jobs API call:

```
curl -u USERNAME:PASSWORD http://proxy-
url:6764/api/jobs/datasource:myDatasource/history
```

The following is returned:

```
[ { "resource" : "datasource:myDatasource", "runId" : "fXfKTQNSKM", "startTime" :
"2020-04-07T22:34:15.852Z", "endTime" : "2020-04-07T22:41:59.271Z", "status" :
"success", "extra" : { "counter.failed" : 0, "counter.other.pipeline.in" : 58242,
"counter.stage.field-mapping::d3ebd921-6065-479e-9d45-0bfed819346b.processed" :
58241, "counter.deleted" : 0, "counter.stage.solr-dynamic-field-name-
mapping::e6aa9c51-2867-40a0-97fd-34d909098a20.processed" : 58241, "counter.new" :
2268, "counter.output" : 58241, "counter.other.pipeline.complete" : 58242,
"datasourceId" : "myDatasource", "counter.input" : 60435, "startTime" :
1586298855843, "endTime" : 1586299313433, "counter.skipped" : 2194,
"counter.stage.solr-index::01e1c561-0440-44b2-afb4-d6ff471f646c.processed" : 58241,
"counter.other.pipeline.out" : 58242 }, "startedBy" : "09765f1d-2f1c-4c3b-ba22-
887e3886d8aa" }, { "resource" : "datasource:myDatasource", "runId" : "XxrxochVOB",
"startTime" : "2020-04-07T20:06:25.819Z", "endTime" : "2020-04-07T20:14:01.686Z",
"status" : "success", "extra" : { "counter.failed" : 0, "counter.other.pipeline.in"
: 59090, "counter.stage.field-mapping::d3ebd921-6065-479e-9d45-
0bfed819346b.processed" : 59089, "counter.deleted" : 0, "counter.stage.solr-
dynamic-field-name-mapping::e6aa9c51-2867-40a0-97fd-34d909098a20.processed" :
59089, "counter.new" : 2268, "counter.output" : 59089,
"counter.other.pipeline.complete" : 59090, "datasourceId" : "myDatasource",
"counter.input" : 61283, "startTime" : 1586289985809, "endTime" : 1586290440433,
"counter.skipped" : 2194, "counter.stage.solr-index::01e1c561-0440-44b2-afb4-
d6ff471f646c.processed" : 59089, "counter.other.pipeline.out" : 59090 },
"startedBy" : "09765f1d-2f1c-4c3b-ba22-887e3886d8aa" }, { "resource" :
"datasource:myDatasource", "runId" : "2jy3UemLTp", "startTime" : "2020-04-
07T19:23:39.576Z", "endTime" : "2020-04-07T19:31:26.694Z", "status" : "success",
"extra" : { "counter.failed" : 0, "counter.other.pipeline.in" : 58906,
"counter.stage.field-mapping::d3ebd921-6065-479e-9d45-0bfed819346b.processed" :
58905, "counter.deleted" : 0, "counter.stage.solr-dynamic-field-name-
mapping::e6aa9c51-2867-40a0-97fd-34d909098a20.processed" : 58905, "counter.new" :
2268, "counter.output" : 58905, "counter.other.pipeline.complete" : 58906,
"datasourceId" : "myDatasource", "counter.input" : 61099, "startTime" :
1586287419556, "endTime" : 1586287883825, "counter.skipped" : 2194,
"counter.stage.solr-index::01e1c561-0440-44b2-afb4-d6ff471f646c.processed" : 58905,
"counter.other.pipeline.out" : 58906 }, "startedBy" : "09765f1d-2f1c-4c3b-ba22-
887e3886d8aa" } ]
```

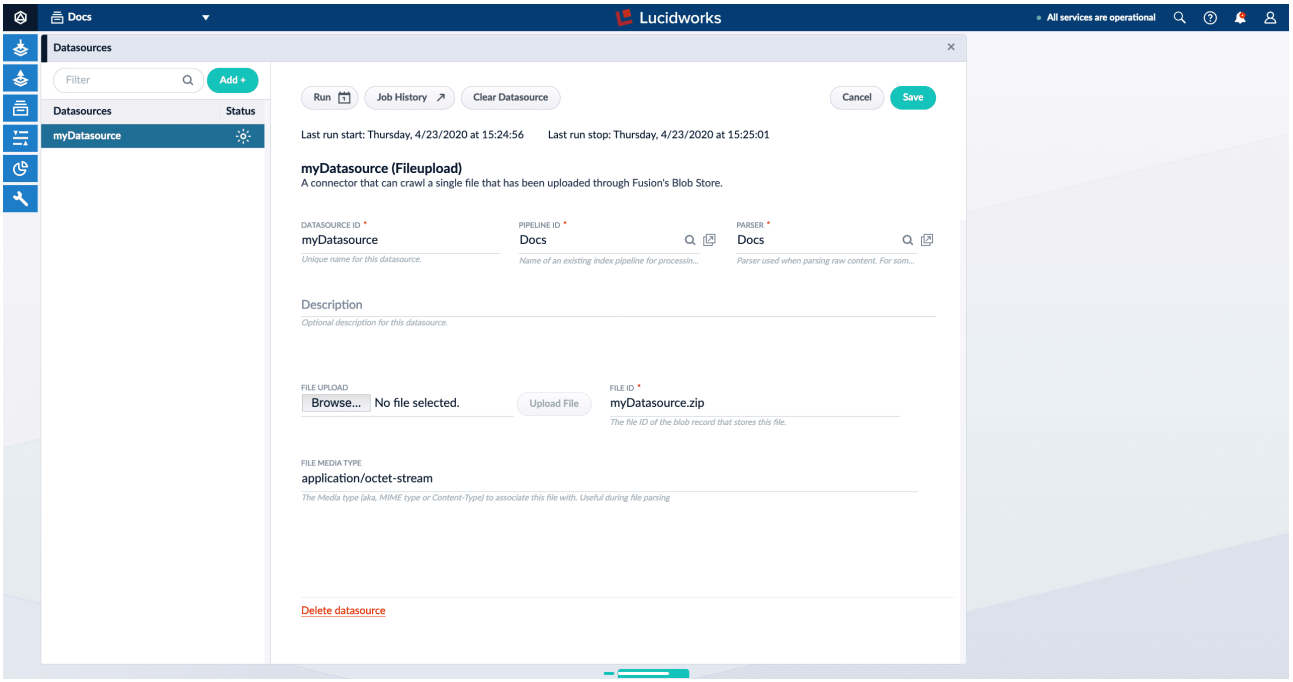

The section *How does the Jobs API work?* below gives additional details on how the Jobs API retrieves information from Fusion. See the *Jobs API reference* for more information.

Job history results field definitions

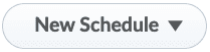
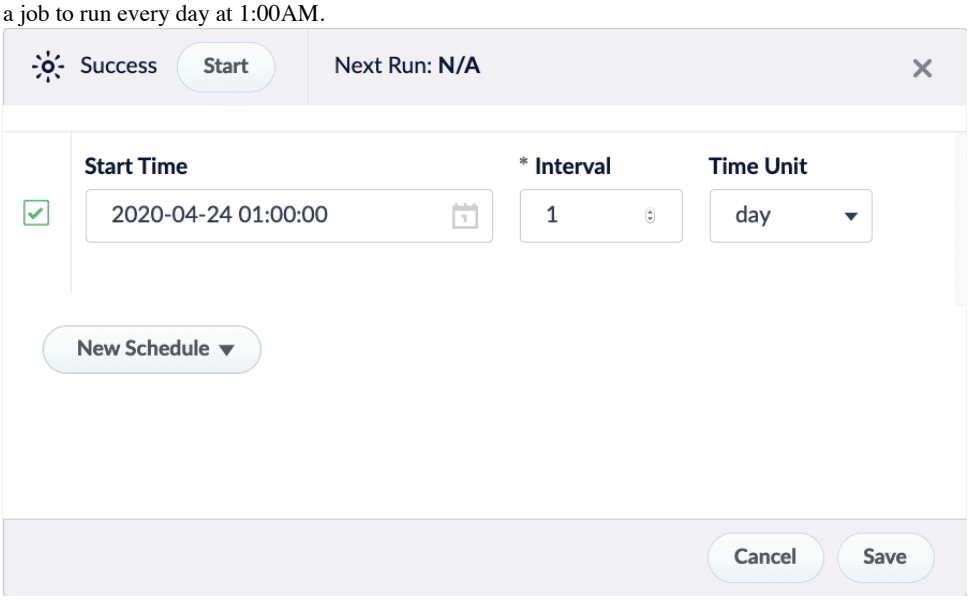
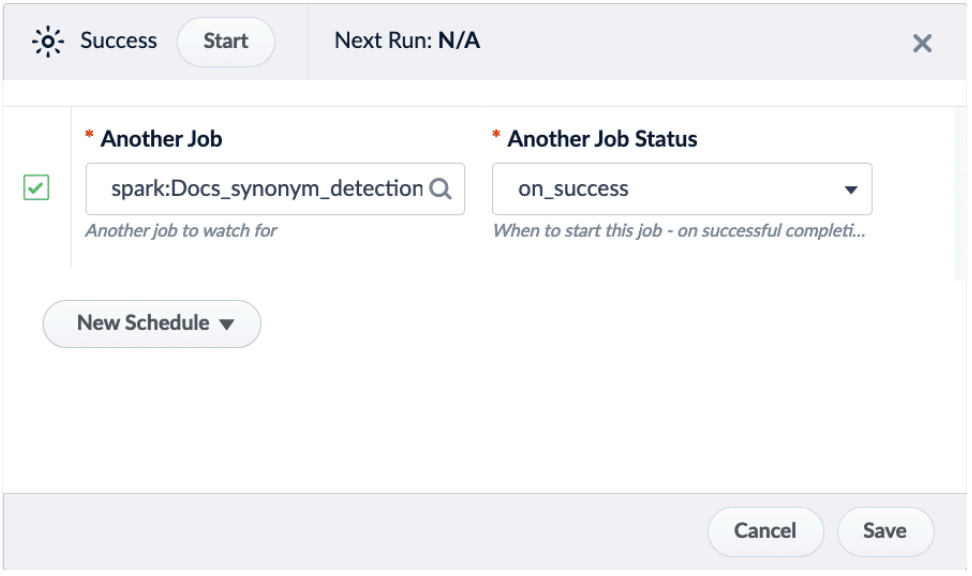
Field	Definition
counter.deleted	Total number of documents deleted or removed from the Solr index.
counter.failed	Total number of documents not indexed due to failure. The errors are written to the log.
counter.input	Total number of fetched documents passed to the index pipeline.
counter.new	Total number of documents fetched for the first time.
counter.output	Total number of documents processed in the run (sum of failed + deleted + indexed documents).
counter.skipped	Total number of documents skipped due to rules such as max, min, and file size.
endTime	Time the job concluded, aborted, or was manually stopped.
resource: datasource	Name of the datasource.
startTime	Time the job started.
status	Job status such as Success, Running, or Aborted.

With the UI

In the Fusion UI, navigate to **Indexing** > **Datasources**. Click a datasource to open the datasource panel.



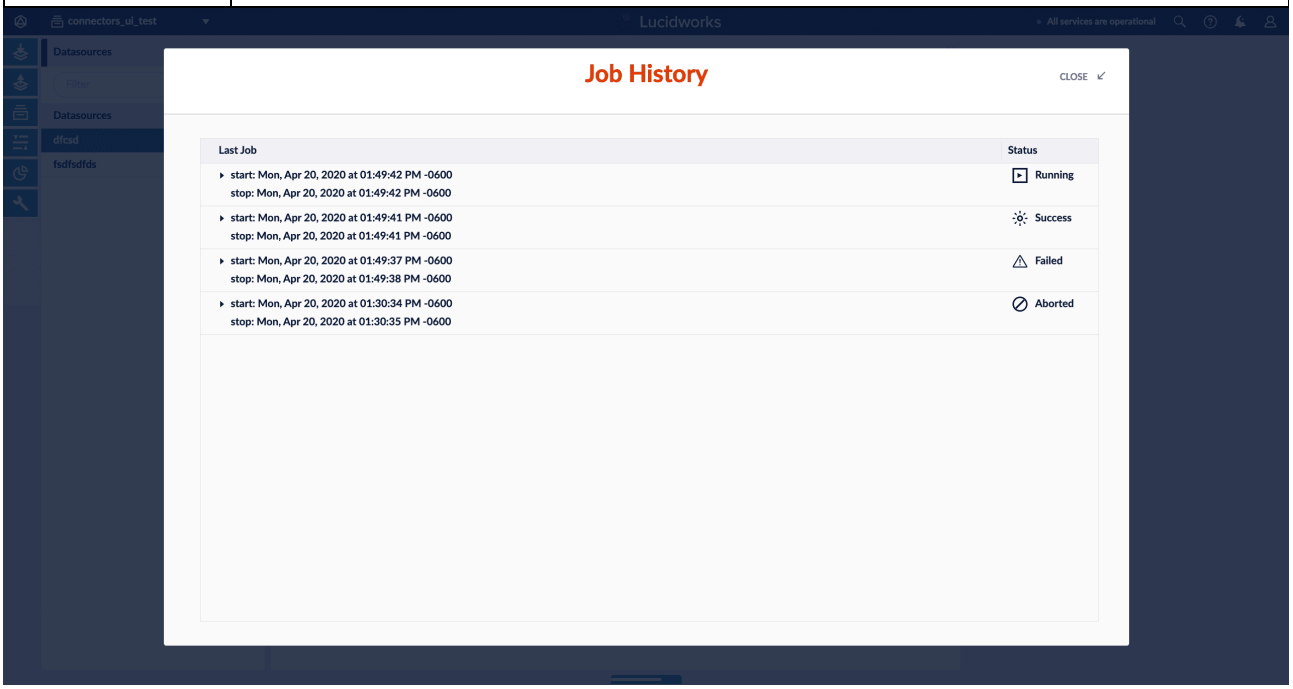
Note that the **Last run start** and **Last run stop** values are equivalent to the `lastStartTime` and `lastEndTime` values from the Jobs API call. (These values are only present if a job has been run at least once.)

Icon	Description
	The **Run** button opens a panel containing various run options and information.
	The **Start** button starts a job.
	The **New Schedule** button displays a dropdown with several scheduling options: • **cron.** Schedule a job based on a Crontab expression. See Cron Trigger documentation *(http://www.quartz-scheduler.org)* for more info. • **interval.** Schedule a job to run on an interval after a set starting point. For example, you can schedule a job to run every day at 1:00AM.  • **job_completion.** Schedule a job to run after another job has completed. You can schedule the job to run if the job succeeds, fails, or regardless of the job's success. 

Icon	Description
Stop	The **Stop** button stops a running job. This results in a status of `"status" : "aborted"`.
Job History ↗	The **Job History** button opens the *Job History panel*. Click one of the job entries to see additional details. Expand the job details for more information. Click **Open Logs in Dashboard** for complete information.
Clear Datasource	The **Clear Datasource** removes documents from your collection that are indexed from that datasource. The job history will *not* be deleted.

In the job history panel, there is an icon on each datasource indicating its latest run history. The possible statuses are:

Icon	Description
Never Run	The job has been created but has not been run.
Running	The job is currently running. Depending on the number of documents being processed, the job can take a long time to complete. Stop the job with the **Stop** button.
Success	The job was completed successfully.
Failed	The job was not able to successfully run. Expand the job details for more information. Click **Open Logs in Dashboard** for complete information.
Aborted	The job was manually stopped before it could finish.



The image above shows all possible job statuses.

How does the Jobs API work?

In *Checking a job status with the API* above, two Jobs API endpoints were used to call the datasource `myDatasource`:

- `myApp/jobs?type=datasource` - Retrieves the current datasource job states for *all* datasources of a specified app. In this case, the app is specified as `myApp`.
  You can also use `type=spark` to retrieve all Spark jobs and `type=task` for task jobs.
- `/jobs/datasource:myDatasource/history` - Retrieves a specific object's job history. In this case, the object type is a datasource, as selected with `datasource:`.
  Datasources are not the only object type for jobs. There are also Spark jobs (`spark:`) and task jobs (`task:`).

Both of the API calls above send a request to Fusion's API application, which serves the Jobs API. The API application contains two main components related to job history data, `JobController` and `SolrJobHistoryStore`.

The Job Controller

The Job Controller is responsible for getting any active job's run status, if an active job is running.

In the case of V1 connectors classic datasource jobs, it will reach out to Fusion's connectors-classic application's connectors API.

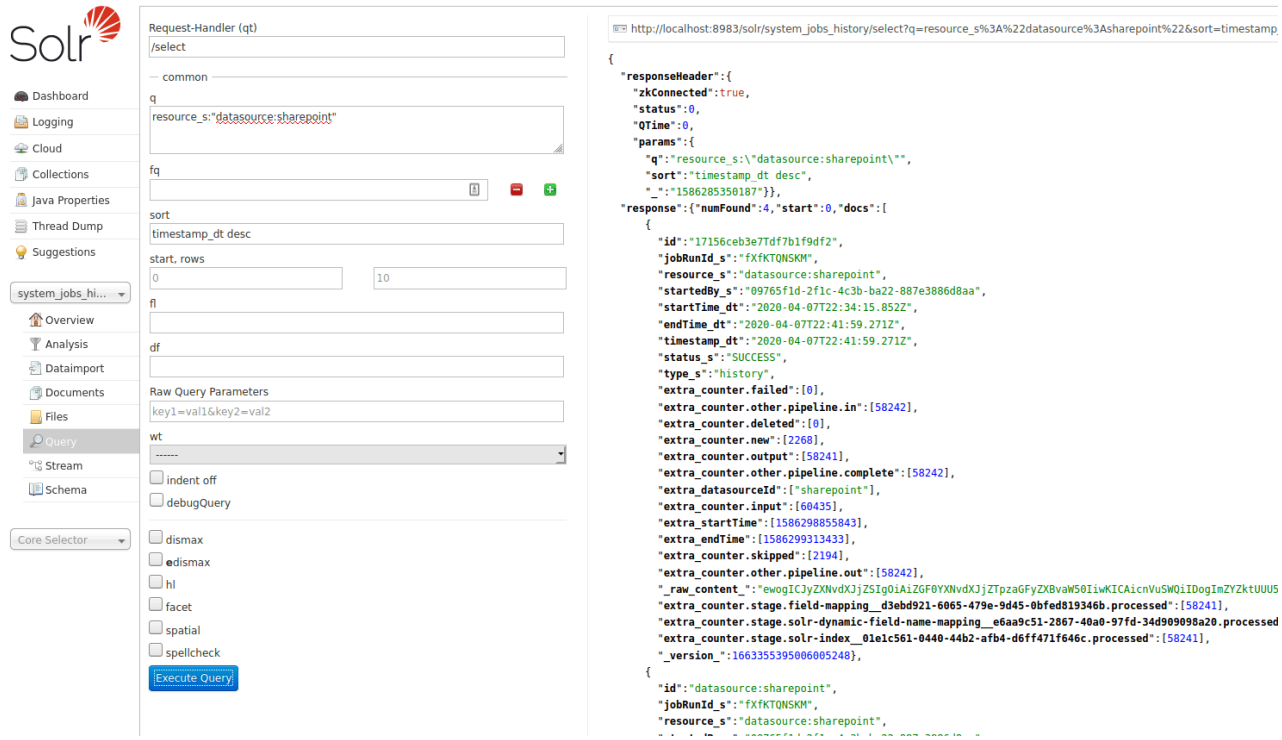
When the API application calls the connectors-classic Jobs API, it will first use Zookeeper to identify the IP address of the connector-classic node that is assigned to this datasource, if one exists.

When the connectors-classic node IP assigned to the job is identified, the API application will call out to the connectors-classic node via `http://<connectors-node-ip>:8984/connectors/v1/connectors/jobs/myDatasource` *(`http://<connectors-node-ip>:8984/connectors/v1/connectors/jobs/myDatasource`)*. The call reaches the component `ConnectorsManagerController` to query the job status of currently running jobs with the ID `myDatasource`.

The Solr Job History Store

`SolrJobHistoryStore` reads and writes a job's history data to and from Solr. Fusion's API application creates a special system Solr collection, `system_jobs_history`, to store the history.

You can also query the job history from Solr directly, if you want:



Spark Jobs

Apache Spark can power a wide variety of data analysis jobs. In Fusion, Spark jobs are especially useful for generating *recommendations*.

Spark job subtypes

For the Spark *job type*, the available subtypes are listed below.

- *SQL Aggregation job*
  A Spark SQL aggregation job where user-defined parameters are injected into a built-in SQL template at runtime.
- *Custom Python job*
  The Custom Python job provides user the ability to run Python code via Fusion. This job supports Python 3.6+ code.
- *Script*
  Run a custom Scala script as a Fusion job.

See *Additional Spark jobs* for more information.

Spark job configuration

Spark jobs can be created and modified using the *Fusion UI* or the *Spark Jobs API*. They can be scheduled using the Fusion UI or the *Spark Jobs API*.

For the complete list of configuration parameters for all Spark job subtypes, see the *Jobs Configuration Reference*.

Machine learning jobs

Fusion provides these job types to perform machine learning tasks.

Signals analysis

These jobs analyze a collection of signals in order to perform *query rewriting*, *signals aggregation*, or *experiment analysis*.

- *Ground Truth*
  Estimate ground truth queries using click signals and query signals, with document relevance per query determined using a click/skip formula.

Query rewriting

These jobs produce data that can be used for query rewriting or to inform updates to the *synonyms.txt file*.

- *Head/Tail Analysis*
  Perform head/tail analysis of queries from collections of raw or aggregated signals, to identify underperforming queries and the reasons. This information is valuable for improving overall conversions, Solr configurations, auto-suggest, product catalogs, and SEO/SEM strategies, in order to improve conversion rates.
- *Phrase Extraction*
  Identify multi-word phrases in signals.
- *Synonym Detection Jobs*
  Use this job to generate pairs of synonyms and pairs of similar queries. Two words are considered potential synonyms when they are used in a similar context in similar queries.
- *Token and Phrase Spell Correction*
  Detect misspellings in queries or documents using the numbers of occurrences of words and phrases.

Signals aggregation

- *SQL Aggregation*
  A Spark SQL aggregation job where user-defined parameters are injected into a built-in SQL template at runtime.

Experiment analysis

- *Ranking Metrics*
  Calculate relevance metrics (nDCG and so on) by replaying ground truth queries against catalog data using variants from an experiment.

Collaborative recommenders

These jobs analyze signals and generate matrices used to provide *collaborative recommendations*.

- *BPR Recommender*
  Use this job when you want to compute user recommendations or item similarities using a Bayesian Personalized Ranking (BPR) recommender algorithm.
- *Query-to-Query Session-Based Similarity jobs*
  This recommender is based on co-occurrence of queries in the context of clicked documents and sessions. It is useful when your data shows that users tend to search for similar items in a single search session. This method of generating query-to-query recommendations is faster and more reliable than the Query-to-Query Similarity recommender job, and is session-based unlike the similar queries previously generated as part of the *Synonym Detection job*.

Content-based recommenders

Content-based recommenders create matrices of similar items based on their content.

- *Content-Based Recommender*
  Use this job when you want to compute item similarities based on their content, such as product descriptions.

Content analysis

- *Cluster Labeling*
  Use this job when you already have clusters or well-defined document categories, and you want to discover and attach keywords to see representative words within those existing clusters. (If you want to create new clusters, use the *Document Clustering job*.)
- *Document Clustering*
  The Document Clustering job uses an unsupervised machine learning algorithm to group documents into clusters based on similarities in their content. You can enable more efficient document exploration by using these clusters as facets, high-level summaries or themes, or to recommend other documents from the same cluster. The job can automatically group similar documents in all kinds of content, such as clinical trials, legal documents, book reviews, blogs, scientific papers, and products.
- *Classification job*
  This job analyzes how your existing documents are categorized and produces a classification model that can be used to predict the categories of new documents at index time.
- *Outlier Detection*
  Use this job when you want to find outliers from a set of documents and attach labels for each outlier group.

Data ingest

- *Parallel Bulk Loader*
  The Parallel Bulk Loader (PBL) job enables bulk ingestion of structured and semi-structured data from big data systems, NoSQL databases, and common file formats like Parquet and Avro.

Tasks

Tasks are a flexible job type that can be used to clean up old logs or run any REST call.

Task subtypes

The "task" job type has the following subtypes:

- *Log Cleanup*
  Delete old log messages from system logs collection.
- *REST Call*
  A versatile job type that runs an arbitrary REST/HTTP/Solr command.

Task configuration

Tasks can be created and modified using the *Fusion UI*. They can be scheduled using the Fusion UI or the *Jobs API*.

See also *Jobs Configuration Reference*.

Security

> Lucidworks recommends *not* virus scanning the `fusion/data` folder. Virus scanning can cause slow performance, and it can cause downtime if it quarantines an index file identified as a possible virus.

Fusion uses a number of security measures:

- **Authenticating UI users** – Fusion authenticates users when they log in. Logging in creates a new Fusion session. Fusion also authenticates users when the *Sessions* REST API creates a session.
- **Authorizing UI users** – Fusion authorizes users to use specific parts of the Fusion UI.
  > UI users must also be authorized to make API requests, because the UI makes API requests.
- **Authenticating and authorizing users who make API requests**
- **Password Encryption**- Fusion uses 128-bit AES keys to encrypt passwords and "AES/CBC/PKCS7Padding" for the cipher. The ciphertext is also signed.
- **Using session cookies**
- **Using an external authentication provider** (optional) - A security realm can specify use of an external authentication provider, such as LDAP, JWT, or SAML.
- **Using SSL/TLS to ensure that data in flight between your application and Fusion is not observable** (optional) - See *SSL Security (Unix)* or *SSL Security (Windows)*.
- **Constraining the documents that are indexed** (optional)
- **Trimming the documents that are returned by queries based on authorization** (optional)

Fusion user login

When logging into the Fusion UI, a user provides a username and password, as well as their assigned security realm. An administrator must specify these in Fusion (using the native security realm) or configure Fusion to use an external authentication provider (for example, LDAP or SAML). See *Access control*.

Fusion uses *roles* defined by *permissions* to authorize Fusion UI access and perform tasks in Fusion, including searching. The recommended method to delegate permissions is as follows:

- Assign each user to a role and create custom roles as needed.
- Assign permissions on a per-app basis.

Manage users with security realms

Fusion uses *security realms* to authenticate users of the Fusion UI. Each user has an assigned security realm, which the user must select when logging in. If the user selects a different realm, authentication fails.

A security realm also provides a list of roles as follows:

- The list always includes the role(s) that are specified in the security realm.
- (Optional) If an external directory service (such as LDAP) is used for authentication, the list can also contain roles that are mapped from the names of the directory-service groups. That is, you can configure a security realm to return group information for users from *that same* directory service.
- (Optional) The security realm can reference one or more Fusion roles or, when using an external directory service provider, use group membership information from the provider to determine roles for users. Fusion maps the group names to role names and adds these roles to the user's list of roles.

> Fusion does not use permissions from LDAP to authorize UI access or API requests. It only obtains group names (optionally), which are used as role names or are mapped to role names. If an *Active Directory Security Query Trimming Stage* is used, then directory-service permissions *are* used for trimming. If a connector supports security trimming, then connector permissions *are* used for trimming.

Per-Request Authentication

Requests to the Fusion REST API must specify a security realm for per-request authentication, unless a session cookie is used (which contains information about the security realm).

Fusion authorizes requested operations based on API permissions specified for the user and for the user's role(s). Fusion considers the role(s) specified in the user definition and in the security realm. Fusion creates a list of roles when a session is created, that is, when a user logs in or when the *Sessions* REST API creates a session. Authorization based on *permissions* and its layering is at request time.

You can define multiple *security realms* for a Fusion instance. A Fusion instance can manage multiple security realms, which allows users from different domains to have (different levels of) access to specific Fusion collections.

Access Control

User Authentication and Authorization

Fusion provides application security by restricting access to known *users* via a two-stage process consisting of:

- **Authentication.** users must sign on using a username and password.
- **Authorization.** each username is associated with one or more permissions which specify the Fusion UI components and REST API requests that user has access to. *Permissions* can be restricted to specific endpoints and path parameters. *Roles* are named sets of permissions which provide access to a specific function.

The access control component runs in the same process as the Fusion UI. It referred to as the "auth proxy" because it handles authentication and authorization for all requests to the Fusion REST API services.

All requests to Fusion must be authenticated, as described in section *User Access Request Params*.

User Account Administration

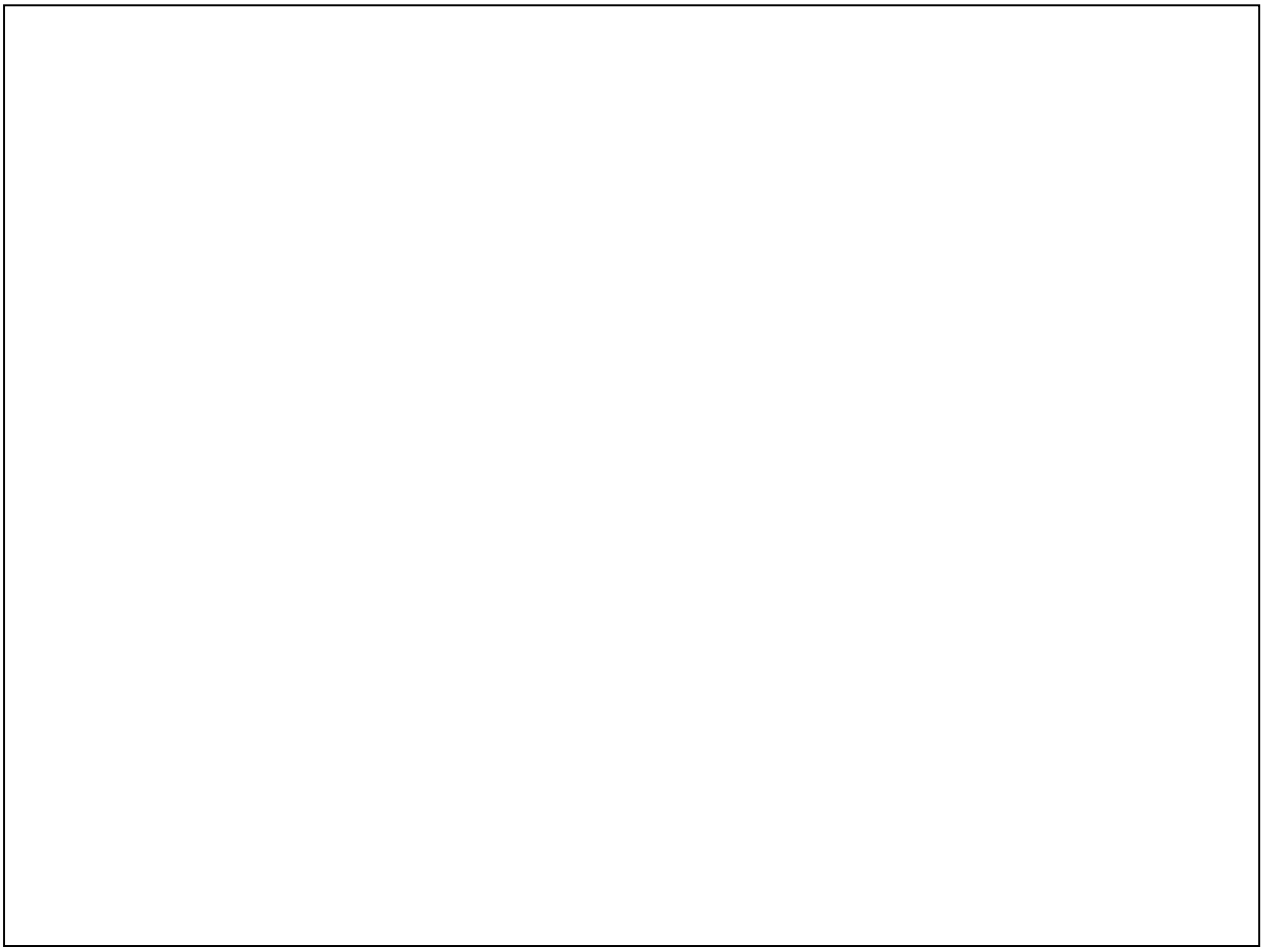
A Fusion *Security Realm* encapsulates a user database together with specific authentication and authorization mechanisms. This information is stored in ZooKeeper *(http://zookeeper.apache.org/)* so that is it always available to all Fusion components across the deployment.

Fusion's native security realm manages both authentication and authorization directly. All user information is stored in ZooKeeper: usernames, hashes of passwords, roles, and permissions. Passwords are hashed using bcrypt *(https://en.wikipedia.org/wiki/Bcrypt)*. Authentication compares a hash of the entered login password with the stored password hash. The native realm is the home of the Fusion admin user and is the default realm type.

Fusion can be configured to use the host domain's security mechanism for user administration. The following configurations are possible:

- **LDAP.** Fusion stores a local user record in ZooKeeper. Authentication is performed by the LDAP server. LDAP group membership can be used to assign Fusion permissions.
- **Kerberos.** Fusion stores a local user record in ZooKeeper. SPNEGO *(https://en.wikipedia.org/wiki/SPNEGO)* is used for authentication via Kerberos.
- **Kerberos authentication and LDAP authorization.** Fusion stores a local user record in ZooKeeper. SPNEGO *(https://en.wikipedia.org/wiki/SPNEGO)* is used for authentication via Kerberos. LDAP group membership can be used to assign Fusion permissions.
- **SAML.** Fusion stores a local user record in ZooKeeper. The SAML 2.0 protocol is used to provide web-browser single sign-on.
- **JWT.** JSON Web token.

Video tutorial



More information

These additional topics explain how to configure the supported authentication methods:

- *Configure Fusion for JWT*
- Configure Fusion for Kerberos in *Unix* and *Windows*
- *Configure Fusion for LDAP*
- *Configure Fusion for SAML*
- *Configure Fusion for SSO*

Permissions

Permissions determine what a user can do in Fusion. There are two kinds of permissions:

- **UI permissions** – Control which parts of the Fusion UI a user can access. These parts show up in menus and the user can view them. But the ability to *use* the functionality depends on API permissions.
- **API permissions** – Control which requests a user can submit to which REST API endpoints.

Permissions can be defined by either a role or a user, or both. Fusion combines permissions for authorization as follows:

- UI permissions are positive (permission needs to be given) and additive (the user has the sum of all specified permissions). This is true of roles specified in a user definition, roles specified in a security realm, and roles determined dynamically based on groups in an LDAP authentication provider.
- API permissions specified in roles are positive (permission needs to be given) and additive (the user has the sum of all specified permissions; that is, for a specific endpoint, the most permissive permissions are used). This is true of roles specified in a user definition, roles specified in a security realm, and roles determined dynamically based on groups in an LDAP authentication provider.
- API permissions specified in the role(s) but not in the user definition are used.
- If an API permission for a specific endpoint is specified in both a role *and* in the user definition, then the permissions in the user definition are used, *overriding* the permissions in the role(s). Use permissions in user definitions to give specific users permissions that are less permissive than the permissions for their role(s).
  For example, say role A allows GET and POST access to a specific endpoint. User X is a member of role A and also has a user definition that allows only GET access to that endpoint. In this case, user X has only GET access to that specific endpoint.
  Alternatively, you could define less permissive roles.

Specify UI Permissions

Specify UI permissions in *roles*.

Specify API Permissions

A Fusion API permission denotes an allowed request to a Fusion REST API endpoint or endpoints. A permissions specification consists of:

- HTTP request method or methods allowed. Multiple HTTP methods are separated by commas.
- REST API services endpoint, which can contain wildcards or named variables. All calls to the REST API start with "api/", followed by the service name and any methods and parameters. The permissions specification includes everything following `api/`. The endpoint can include wildcards.
  Wildcards make it easy to grant broad access to Fusion services. The wildcard symbol `*` matches all possible values for a single path segment. Two wildcards match all possible values for any number of path segments. Granting access to a subset of Fusion functionality can require a list of narrowly defined permissions. Carefully defining each Fusion app to address a specific use case can simplify permission sets, by letting you grant access liberally within a single app.
  A path segment can be a named variable enclosed in curly braces: `{variable-name}`. Variables are used when a wildcard would be too permissive and a single path segment too restrictive.
- Optionally, the allowed values for any named variables in the endpoint. The variable specification component specifies the restricted value or values for all named variables in the path. Each specification consists of the variable name, followed by "=" (the equals sign), followed by one or more values separated by commas. If the endpoint specification has multiple variable, the semi-colon character ";" is used as the separator between parameter specifications.

A permission specification has three elements: methods, path, and parameters. A permissions specification is written as a string using the colon character ":" as the separator between the elements. The lists of permission specifications are stored in the ZooKeeper User and Roles entries in JSON format.

Example Permission Sets

Admin permissions

The permissions for the admin user can be written in a single line:

```
GET,POST,PUT,DELETE,PATCH,HEAD:/**
```

This permission specification uses two wildcards. One wildcard specifies any path segment, and the other specifies any value.

Best practice: Delegate access to a Fusion app

When possible, create a Fusion app for a specific use case and assign permissions on a per-app basis. For example, to configure a Fusion account to be able to run queries against only a certain collection:

1. *Create a Fusion app*.
2. *Create a query profile* by defining a collection. This lets you change the underlying collection without having to redo permissions.
3. Assign permissions as follows:

```
GET:/apps/APP_NAME/query/QUERY_PROFILE/*
```

The above permission set allows a user to run queries against the collection specified by `QUERY_PROFILE` under the Fusion app `APP_NAME`.

Roles

Roles are named sets of *permissions* that encapsulate the permissions needed for different kinds of users. Permissions grant users access to subsets of Fusion functionality. A role can specify UI permissions, API permissions, or both:

- UI permissions grant users access to parts of the Fusion UI
- API permissions grant users access to specific API commands for specific REST API endpoints.

See *Permissions* for information about how permissions supplied by multiple roles and by user definitions combine.

	Users who can create or modify code obtain access to the broader Fusion environment. This access can be used to create intentional or unintentional damage to Fusion.

Where You Specify Roles

You can specify which roles to apply for a user in one or more of these places:

- **Security realm (directly)** – Under the heading **Roles**, specify the roles to always apply to all users in the security realm.
- **Security realm (from a group/role mapping)** – Security realms of types `ldap` and `trusted-http` can provide a list of groups to which the user belongs. The security realm can map the group names to role names.
- **User definition** – A user definition can specify roles for the user. These roles do not override the other roles. They are added to the other roles.

Default Roles

At initial startup, Fusion creates a set of default roles for common types of users.

admin

The admin role is the equivalent to the Unix `root` or superuser. It allows full access to all Fusion services:

```
GET,POST,PUT,DELETE,PATCH,HEAD:/**
```

developer

The developer role has all the read/write permissions required for building and running applications.

```
GET,POST,PUT:/system/** GET,POST,PUT,DELETE,HEAD:/stopwords/**
GET,POST,PUT:/usage/** GET:/features/** GET,POST,PUT,DELETE,HEAD:/blobs/**
GET,POST,PUT,DELETE,HEAD:/scheduler/** GET,POST,PUT,DELETE,HEAD:/experiments
GET:/introspect/** PUT:/usage/** GET,POST,PUT,DELETE,HEAD:/index-stages/**
GET,POST,PUT,DELETE,HEAD:/messaging/** GET,POST,PUT,DELETE,HEAD:/catalog
GET,POST,PUT,DELETE,HEAD:/parsers/** GET,POST,PUT:/appkit/**
GET,POST,PUT,DELETE,HEAD:/index-profiles/** GET,POST,PUT:/recommend/**
GET,POST,PUT,DELETE,HEAD:/history/** GET,POST,PUT,DELETE,HEAD:/apps/**
GET,POST,PUT,DELETE,HEAD:/solr/** GET,POST:/query/** GET,POST,PUT:/signals/**
GET,POST,PUT:/searchLogs/** GET,POST,PUT,DELETE,HEAD:/query-pipelines/**
GET,POST,PUT:/configurations/** GET:/suggestions/**
GET,POST,PUT,DELETE,HEAD:/searchCluster/** GET,POST,PUT,DELETE,HEAD:/index-
pipelines/** GET:/license GET,POST,PUT,DELETE,HEAD:/spark/**
GET,POST,PUT,DELETE,HEAD:/query-stages/**
GET,POST,PUT,DELETE,HEAD:/prefs/apps/search/* GET:/nodes/**
GET,POST,PUT,DELETE,HEAD:/solrAdmin/** GET,POST,PUT:/synonyms/**
GET,POST,PUT,DELETE,HEAD:/jobs/** GET,POST,PUT,DELETE,HEAD,OPTIONS:/collections/**
GET,POST,PUT,DELETE,HEAD:/connectors/** GET,POST,PUT,DELETE,HEAD:/groups/**
GET,POST,PUT,DELETE,HEAD:/query-profiles/** GET,POST,PUT:/templates/**
GET,POST,PUT,DELETE,HEAD:/tasks/** GET,POST,PUT,DELETE,HEAD:/links/**
PATCH:/users/{id}:id=#ID GET,POST,PUT:/registration/** POST:/index/**
GET,POST,PUT:/objects/** GET,POST,PUT:/templates/**
```

The permission `PATCH:/users/{id}:id=#ID` uses the variable value `#ID` as a placeholder for the currently logged-in user ID. It is included so the Fusion UI "change password" feature is available to native realm users.

rules

The rules role provides *query rewriting* API access to all Fusion apps.

```
GET:/apps/*/query-profiles/** GET,POST,PUT,PATCH,DELETE,HEAD:/apps/*/query-
rewrite/** GET:/solr/** GET:/query/** GET:/collections/** GET:/apps/**
```

search

The search role has read-only query and write-only signal API access to the Fusion "default" collection. These permissions are required for search applications, for example, for App Studio.

```
POST:/apps/*/signals/** GET,POST:/query/** POST:/signals/**
PATCH:/users/{id}:id=#ID GET,POST:/apps/*/query/**
```

The permission `PATCH:/users/{id}:id=#ID` uses the variable value `#ID` as a placeholder for the currently logged-in user ID. It is included so the Fusion UI "change password" feature is available to native realm users.

webapps-role

The webapps role can list and download Fusion apps.

```
GET,HEAD:/webapps/** GET,HEAD:/license
```

Role Information

Fusion stores role information in Apache ZooKeeper *(http://zookeeper.apache.org/)*. Each role in a ZooKeeper entry contains the following:

- `id` – ID string, created by Fusion
- `name` – Role name string
- `desc` – Text description; optional
- `permissions` – A list of Fusion permission specifications
- `ui-permisions` – A list of names of Fusion UI components
- `created-at` – Timestamp; created by Fusion
- `updated-at` – Timestamp for last edit; created by Fusion

Manage Roles

Only Fusion users with admin privileges can manage roles.

Restricting access to a subset of Fusion's functionality requires several narrowly defined permissions. Path variables can be used to designate specific collections. As an example, it is possible to define a role which allows read-only access to Fusion dashboards for a specific collection:

- `GET:/solr/{id}/*:id=test` – Read-only access to the collection "test"
- `GET:/solr/{id}/admin/luke:id=test` – Also read-only access
- `GET:/solr/system_banana/*` – Read-only access to dashboards
- `GET:/collections/system_banana` – Read-only access to the collection where dashboard definitions are stored

Manage Roles in the Fusion UI

Manage roles in the Fusion UI. Click **System > Access Control > Roles**.

To create a new role from the Fusion UI, first choose a unique role name, then edit the set of permissions. Specify API permissions one per line in the **Permissions** input box. There is a separate list of checkboxes that allow access to the Fusion UI components. If users who are assigned this role require access to the Fusion UI, then you must specify UI permissions in addition to REST API permissions.

Manage Roles via HTTP Requests to the Roles API

See page *Roles API*.

Security Realms

Fusion uses *security realms* to authenticate users of the Fusion UI. Each user has an assigned security realm, which the user must choose when logging in. Choosing a different realm results in an authentication failure.

A security realm also provides a list of roles:

- The list always includes the role(s) that are specified in the security realm.
- (Optional) The security realm can reference one or more Fusion roles and/or get groups to which the user belongs from an external directory service that is the authentication provider. Fusion maps the group names to role names and adds these roles to the user's list of roles.

	Fusion does not use permissions from the LDAP to authorize UI access or API requests. It only obtains group names (optionally), which are used as role names or are mapped to role names. If an *Active Directory Security Query Trimming Stage* is used, then directory-service permissions *are* used for trimming. If a connector supports security trimming, then connector permissions *are* used for trimming.

Requests to the Fusion REST API must specify a security realm for per-request authentication, unless a session cookie is used (which contains information about the security realm).

Fusion authorizes requested operations based on API permissions specified for the user and for the user's role(s). Fusion considers the role(s) specified in the user definition and in the security realm. Fusion creates a list of roles when a session is created, that is, when a user logs in or when the *Sessions* REST API creates a session. Authorization based on permissions is at request time.

You can define multiple security realms for a Fusion instance. This lets you give different sets of users different levels of access to specific Fusion collections.

	The directory service that you use to return group information must be the same one that is used for authentication.

Security Realm Types

When you create a security realm, you can choose among the following security realm types:

OpenID Connect

OpenID Connect is an identity authorization layer which supplements the OAuth 2.0 protocol.

	Fusion's OpenID Connect security realm has been tested with Google *(https://developers.google.com/identity/protocols/OpenIDConnect)* and Okta *(https://developer.okta.com/docs/reference/api/oidc)*.

See *Configure Fusion for OpenID Connect*.

Native

Fusion has a single preconfigured security realm named *native*. The admin user is in the native realm. The native realm also provides a fallback mechanism in case of LDAP server or communication failure.

This realm is required to bootstrap Fusion. Because all requests to Fusion require authentication and authorization, on initial startup you must access the Fusion UI to set the admin password. After Fusion has a valid admin password, it creates the admin account in the Fusion native realm.

For the native realm, Fusion manages all authentication and permissions information directly.

You can create Fusion user accounts and manage them using either the Fusion UI or the *User API*.

Passwords are not stored, but are hashed using bcrypt *(https://en.wikipedia.org/wiki/Bcrypt)*.

SSO Trusted HTTP

The "SSO Trusted HTTP" realm type (`trusted-http` in the REST API) is useful in single sign-on (SSO) environments.

If the SSO environment contains groups that make sense regarding partitioning Fusion functionality for Fusion users (that is, giving Fusion users different UI and API permissions based on the SSO groups), then you can configure an SSO Trusted HTTP security realm to return a list of group names, and then map the groups to Fusion roles in the security realm definition.

See *Configure Fusion for SSO*.

JWT

The JSON Web Token (JWT) realm uses an "Authorization" header in the request to authenticate the user and the data inside the JWT token for the authorization. You can configure Fusion to use a shared secret key to encrypt the JWT payload.

For Fusion's JWT realm, first create a JWT token, with the tool you use to validate users. Then create a Fusion realm.

See *Configure Fusion for JWT*.

Kerberos

In the case where a host domain uses Kerberos for authentication and LDAP for authorization, Fusion can be configured to do the same, by configuring a realm of type "LDAP" and then specifying Kerberos as the authentication mechanism.

Fusion stores a local user record in ZooKeeper and a mapping to the Kerberos principal *(https://docs.oracle.com/cd/E21455_01/common/tutorials/kerberos_principal.html)*.

SPNEGO *(https://en.wikipedia.org/wiki/SPNEGO)* is used for authentication via Kerberos.

See how to configure Fusion for Kerberos in *Unix* and *Windows*.

LDAP

You can use LDAP as an authentication provider for Fusion. If the LDAP contains groups that make sense regarding partitioning Fusion functionality for Fusion users (that is, giving Fusion users different UI and API permissions based on the LDAP-group memberships of LDAP users), then you can configure an LDAP security realm to search for LDAP groups and to map the LDAP groups to Fusion roles.

Fusion stores a local user record in ZooKeeper, and authentication is performed by the LDAP server. User accounts can be managed by Fusion, or created automatically, in which case the Fusion user ID maps directly to the LDAP Distinguished Name *(https://en.wikipedia.org/wiki/Lightweight_Directory_Access_Protocol#Directory_structure)* (DN). Fusion permissions can be assigned automatically based on LDAP group membership.

See *Configure Fusion for LDAP*.

SAML

Fusion stores a local user record in ZooKeeper and the URL and information about the SAML Identity Provider. The SAML 2.0 protocol is used to provide web browser single sign-on.

See *Configure Fusion for SAML*.

Manage Security Realms

Only Fusion users with *admin privileges* can manage security realms. There are two ways to manage security realms:

In the Fusion UI
Navigate to **System** > **Access Control** > **Security Realms**.

Using the Realms API
Use the `http://FUSION_HOST:FUSION_PORT/api/realm-configs/` endpoint to manage security realms. See the *Realms API* reference for details. In production environments, use port 8765.

Users

All Fusion requests must come from a registered user.

Add Users

The first user who logs in becomes the user `admin`.

There are two approaches for adding users:

- **Manual** – Add users manually to a security realm that does not auto-create users.
- **Automatic** – For a security realm that uses an external authentication provider, Fusion can add users automatically. When creating the security realm, check **auto-create users**. Fusion creates a user the first time someone logs into Fusion.

When you add a new user manually, you must provide a unique username and valid password. All other information is optional. However, unless either roles or permissions are specified (or both), this user will not be able to do anything in Fusion.

If you specify API permissions in a user definition, those permissions *override* corresponding permissions defined in the user's roles. See *Permissions* for more information about how permissions supplied by multiple roles and by user definitions combine.

Manage Users in the Fusion UI

Only Fusion users with administrative privileges (for example, those who are assigned the built-in role `admin`) can manage users.

Manage users in the Fusion UI. Click **System > Access Control > Users**.

Manage Users via HTTP Requests to the Users API

See page *Users API*.

User Information

Fusion stores user information in Apache ZooKeeper *(http://zookeeper.apache.org/)*.

Each User entry in ZooKeeper contains the following:

- `id`– A globally unique user ID (UUID), created by Fusion based on username, realm-name
- `realm-name`– The Fusion *security realm* name; the default is "native".
- `username`– The username string, which is unique within the specified security realm
- `permissions`– List of permissions that have been explicitly assigned to the user in the Fusion UI (in **System > Access Control**)
- `role-names`– List of roles assigned to the user in the Fusion UI (in **System > Access Control**)
- `created-at`– Timestamp; created by Fusion
- `updated-at`– Timestamp for the last edit; created by Fusion

The following JSON shows the ZooKeeper record for the Fusion admin user:

```
{ "id":"57f539d2-3f53-4011-ad6f-257a3f00fc6b", "username":"admin", "realm-
name":"native" "password-
hash":"$2a$08$3I82umlXLPSshQIW6ngj.Or06DOVgDLGohGmCB9GC0yRtvy5Nfkn6",
"permissions":[], "role-names":["admin"], "created-at":"2016-01-28T00:00:18Z" }
```

The following JSON shows the ZooKeeper record for a user entry managed by Fusion:

```
{ "id":"ae9b345a-79e2-4e6d-8620-e6ed4ed2cc16", "username":"firstname.lastname",
"realm-name":"lwLDAP", "permissions":[{"path":"collections/**","methods":["GET"]}],
"role-names":[], "created-at":"2016-04-01T21:17:36Z" "updated-at":"2016-04-
01T21:42:15Z", }
```

High Availability and Disaster Recovery

High availability

High system availability is sustained by sufficient levels of redundancy in the processing stack. Factors to consider when determining system availability include:

- Redundant resource costs
- System performance
- System synchronization methods and the level of synchronization each method provides
- Computational costs
- Network latency

Redundancy levels available are:

- **Data Redundancy.** Create multiple copies of the data on a single storage device to increase the probability of at least one copy being available at all times.
- **Storage Redundancy.** Distribute the data across multiple physical storage devices to ensure data is available if one of the physical storage devices is not accessible.
- **Process/Pod Redundancy.** Create multiple copies of each service/pod to reduce the risk of a service becoming unavailable if an individual copy fails for some reason.
- **Process/Pod Distribution.** Distribute duplicate pods across different nodes in a cluster to protect against the loss of service due to a failed node in the cluster.
- **Node Redundancy.** Create multiple nodes in the cluster within a single Availability Zone to protect against the loss of any single node. For this option:
  - The level of node over-provisioning is a function of both the probability of a node failing, and the average time required to provision and return a new node to service after a failure.
  - Sufficient redundancy means that if one node fails, the remaining nodes have sufficient computing capacity to manage the load while a new node is provisioned.
- **Availability Zone Redundancy.** Distribute resources across multiple zones to allow the system to tolerate the loss of an entire zone without compromising the availability of the system as a whole. While this is an option, Lucidworks does not use this because it does not consider this best practice.
- **Cluster Redundancy.** - Create multiple copies at the cluster level within a single data center to ensure that there is no loss of service even if an entire cluster is lost.
- **Data Center Redundancy.** Distribute clusters across multiple data centers in a region. If availability to a specific data center is lost, the system is still available in other data centers.
- **Region Redundancy.** Distribute clusters across multiple regions, for example, east and west. If a major outage occurs and availability to a specific region is lost, the system is still available in the other regions.

Query runtime options

The following options describe Fusion configurations.

> The options are listed in order, from the least highly-available and lower disaster-prepared state, to the optimal highly-available and highest disaster-prepared state.

- **Single cluster located in on-premise client hardware.** This provides the lowest level of availability and disaster recovery if an outage or data corruption occurs.
- **Single clusted deployed in the cloud.** This provides the lowest cloud-based level of availability and disaster recovery if an outage or data corruption occurs.
- **Active-Passive disaster recovery.** Redundant clusters exist, but only one is running at a given time.
- **Green-Blue-Active-Passive disaster recovery.** Redundant clusters exist, but only one is available to service user requests.
- **Active-Active disaster recovery.** Redundant clusters exist that contain identical live infrastructure and all clusters are actively servicing user requests. For more information about the configurate to synchronize indexes between data clusters, see *Active - Active*.
- **Green-Blue Active-Active disaster recovery.** Redundant clusters exist that contain duplicate Kubernetes-deployed environments and all clusters are actively servicing user requests.
- **Fully Active-Active Traffic Shaped disaster recovery.** - Redundant clusters exist and all clusters are actively servicing user requests, with traffic directed to a specific cluster according to geography or user affinity.

Fusion High Availability architecture types

Architecture types depend on the number of nodes in a cluster.

Nodes in cluster	Services on node
All services on 3+ nodes	• All services run on each node.
All services on 2+1 nodes	• All services run on each node. • Requires an additional node to run only Zookeeper.
Deploy services separately on 10 nodes	• 3 nodes for only search services (admin-ui, api, Zookeeper) • 2 nodes for only Solr service (search engine) • 2 nodes for signals services (Spark master and worker) • 2 nodes for connectors services

Fusion Disaster Recovery deployment architecture types

Disaster Recovery uses independent Fusion clusters on both data centers.

The key tasks are to synchronize:

- data center configurations
- the actual data (indexes)

Strategy to sync data center configuration

Extract configurations from lower environment and move to higher environment. During this process we ensure configurations apply to high environments for both data centers.

Setups to sync indexes between data centers

Active - Active

1. **Hot - Hot**: The data centers connect to a common node for good network bandwidth.
   a. Fusion clusters are on both data centers without Solr.
   b. Deploy Solr service on the common node.
   c. Both Fusion clusters connect to Solr node to GET/PUT catalog or signals records.
   d. Solr index has its own backup and restore procedure.
2. **Hot - Warm:** There is no common data node between the data centers.
   a. Deploy Fusion full stack clusters on both data centers.
   b. Use Fusion connectors or PBL job to transfer catalog and signals from one datacenter to another.
   c. Solr index is already replicated on both data centers.
   d. During data transfer, it is possible that data is not fully synchronized.

Active - Passive

1. **Hot - Cold**
   a. Fusion has 1-2 nodes on the secondary datacenter.
   b. Configurations are periodically copied over to the secondary data center.
   c. Connectors job is set up to copy indexes.
   d. Could also take periodic Solr Active cluster backups and cache them temporarily so the most recent backup is always available to restore into the Cold cluster when it starts.

Data transfer options

Fusion SolrIndex connector

For the *Fusion "SolrIndex" Connector*:

- Pros include:
  - Easy to configure.
  - Batch transfer, but can be as fast as streaming using scheduling and monitoring options.
  - Configurable to filter out documents.
- Cons include:
  - Not applicable if the data transfer rate requirement is too high.

- Only to transfer less than 100 million documents.

Fusion Parallel Bulk Loader (PBL) job

For the *Fusion Parallel Bulk Loader (PBL) job*:

- Pros include:
  - Easy to configure Fusion Spark job.
  - Blazing fast batch transfer with scheduling and monitoring options available.
  - Configurable to filter out documents.
- Cons include:
  - None, because this option does not need to run Spark services.

Solr streaming expressions

Here is an example of Solr streaming expressions:

```
daemon(id="uniqueId", runInterval="5000", terminate="false", commit(Target,
batchSize=5000, update(Target, batchSize=500, topic( Source-Checkpoint-Coll,
Source, zkHost="localhost:9983/solr", q="*:*", fq="mimeType_s:application/pdf",
fl="id,*", sort="id asc", qt="/export" ) ) ) )
```

- Pros include:
  - Keeps track of what has been read to date and only sends what is new.
  - Open source and easy to configure.
  - Does not depend on shard or replica count, and will tolerate differences on either end of the transaction.
  - Tolerates source-side outages and picks up where it left off.
- Cons include:
  - Can only send stored fields.
  - Not parallelizable like Spark jobs.
  - Does not tolerate destination-side outages. When the target is not available, data transfers can drop because the source-side high-water-mark may move before the target-side failure is detected.

Ingress and Security

The Fusion proxy service serves as an API gateway and provides authentication and authorization. Route all external access to Fusion services through the Fusion proxy service. The most common approach is to set up a Kubernetes Ingress *(https://kubernetes.io/docs/concepts/services-networking/ingress/)* that routes Fusion service requests to the proxy service as shown in the example ingress definition *below*. It's also common to do TLS termination *(https://cloud.google.com/load-balancing/docs/https/#tls_support)* at the Ingress, so that all traffic to and from the Kubernetes cluster is encrypted.

Internal requests happen over unencrypted HTTP.

```
apiVersion: v1 items: - apiVersion: extensions/v1beta1 kind: Ingress metadata:
annotations: ... labels: ... name: <RELEASE>-api-gateway namespace: <NAMESPACE>
spec: rules: - host: <HOSTNAME> http: paths: - backend: serviceName: proxy
servicePort: 6764 path: "/*" tls: - hosts: - <HOSTNAME> secretName: <RELEASE>-api-
gateway-tls-secret status: loadBalancer: ingress: - ip: <SOME_IP>
```

EKS

For deployments that run on EKS, the setup_f5_eks.sh *(https://github.com/lucidworks/fusion-cloud-native/blob/master/setup_f5_eks.sh)* setup script creates a classic elastic load balancer (ELB) to expose the Fusion proxy service for demonstration purposes. However, you're strongly encouraged to configure a Kubernetes Ingress with TLS termination in front of the proxy service. See How do I terminate HTTPS traffic on Amazon EKS workloads with ACM? *(https://aws.amazon.com/premiumsupport/knowledge-center/terminate-https-traffic-eks-acm/)* for more information.

You can also opt to use an Application Load balancer *(https://github.com/lucidworks/fusion-cloud-native#eks-ingress)* instead of the proxy service.

AKS

AKS deployments can use the setup_f5_aks.sh *(https://github.com/lucidworks/fusion-cloud-native/blob/master/setup_f5_aks.sh)* setup script, which exposes the Fusion proxy service on an external IP using HTTP for demonstration purposes. We recommend that you configure a Kubernetes Ingress with TLS termination *(https://github.com/lucidworks/fusion-cloud-native#aks-ingress)* in front of the proxy service.

Other Kubernetes Platforms

If your deployment runs on OpenShift using a cluster created with OpenShift Infrastructure Provider, the setup_f5_k8s.sh *(https://github.com/lucidworks/fusion-cloud-native/blob/master/setup_f5_k8s.sh)* setup script will install Fusion 5 on your existing cluster. OpenShift Online services are not supported.

See the RedHat OpenShift *(https://github.com/lucidworks/fusion-cloud-native#redhat-openshift)* section of the readme for more tips toward a successful installation.

If your deployment runs on GKE, the setup_f5_gke.sh *(https://github.com/lucidworks/fusion-cloud-native/blob/master/setup_f5_gke.sh)* setup script in the `fusion-cloud-native` repo provides the option to create the Ingress and TLS cert *(https://github.com/lucidworks/fusion-cloud-native#gke-ingress-and-tls)*.

This process uses Let's Encrypt *(https://letsencrypt.org/)*.

For other Kubernetes providers, refer to provider's documentation on creating an Ingress and TLS certificates.

Monitoring

Fusion has functions built in for observability and monitoring with Fusion's tools or third-party monitoring tools.

Fusion system metrics

Fusion system metrics are logged and available in the UI, in log files, and through the REST API.

- For information about how to monitor, troubleshoot, and investigate incidents, see *The DevOps Center*.
- Read about *System Admin APIs*, including APIs for metrics and cluster status, in the REST API Reference Guide.

Grafana Dashboards

Lucidworks provides several Grafana dashboards designed for Fusion.

Dashboards

Pipeline and Stage Metrics

Both the Indexing and Query service provide per-pipeline, per-stage execution times for the 99th, 95th, 75th, and 50th percentiles. These are commonly used to determine where there may be a performance bottleneck when pipelines are executing slowly.

Indexing Service Dashboard

The Indexing Service has its own Grafana dashboard, which highlights key performance metrics for understanding the behavior of the Fusion 5 Indexing pipelines. It provides per-pipeline and per-stage execution time metrics, along with metrics for document indexing.

Query Service Dashboard

The Query Service also has its own Grafana dashboard that shows the performance of the pipelines and query stages that are being run. It shows the global and per-pod request metrics, in addition to per-pipeline query stage execution times.

Gateway Metrics Dashboard

The Gateway Metrics provides visibility into the behavior of the API Gateway, which stands in front of Fusion 5 services. It includes global request rate gauge, in addition to per-handler and per-route request metrics. This dashboard can be used for locating service slowdowns, as it can highlight particular routes or handlers that are responding slowly.

JVM Metrics Dashboard

The JVM Metrics Dashboard provides information about the CPU and memory utilization of Fusion 5 services, along with some GC metrics. These can be used to assess the JVM performance of Fusion 5 microservices and may help locate bottlenecks, such as being CPU-pegged or hitting frequent garbage collection events that pause the application. Issues relating to garbage collection can often be solved by modifying the JVM options of the process and can then be monitored with this dashboard.

Solr Dashboards

Solr has its metrics split into three distinct dashboards, each covering a separate area of concern for monitoring Solr performance.

Core Dashboard

The Solr Core dashboard contains metrics by the associated Solr core, including per-core metrics for requests, file system utilization, document processing and access, and handler usage.

Node Dashboard

The Solr Node dashboard contains per-node metrics including information about requests, errors, cores, thread pool utilization, and connections. Per-node metrics are useful for tracking down issues with a particular Solr node, especially when these issues do not affect all nodes.

System Dashboard

The Solr System Dashboard shows high-level system stats, including requests/response statistics, JVM metrics, and common operating system metrics, such as memory usage, file descriptor usage, and cpu utilization.

Kubernetes Dashboards

The set of Kubernetes performance dashboards requires the installation of additional daemonsets to collect performance metrics from Kube.

Ensure that at least one namespace in your cluster uses the following configuration in the Prometheus `values.yaml` file for the `stable/prometheus` Helm chart.

```
kubeStateMetrics: enabled: true nodeExporter: enabled: true
```

Additionally you will want to install cadvisor (likely into an infrastructure/system namespace):

```
helm repo add code-chris https://code-chris.github.io/helm-charts helm repo update
helm install cadvisor code-chris/cadvisor
```

Kube Metrics Dashboard (replaced by Kube Node Dashboard)

There are Kubernetes metrics that can be queried through Grafana. However, these require installing a daemonset when setting up Prometheus, which displays information about running pods, memory allocation, and CRON jobs.

Kube Node Dashboard

There are Kubernetes metrics that can be queried through Grafana. However, these require installing a daemonset when setting up Prometheus, which displays information about running pods, load, network traffic, memory allocation and more. Some dashboards may not be rendered until you drill down into one or more specific nodes/instances at the top of the dashboard.

Kubernetes Persistence Volumes Dashboard

This dashboard shows you all of your Kubernetes PVCs, filterable by namespace and/or node. It provides you with information about disk utilization so you can monitor for PVCs that are nearing capacity.

Docker Host and Container Overview Dashboard

This dashboard provides you a great view of your running containers, filterable by namespace, node, and/or container name. It provides information about number of running containers, network traffic, memory utilization, disk IO, and load metrics.

Downloads

- Indexing Service *(https://raw.githubusercontent.com/lucidworks/fusion-cloud-native/master/monitoring/grafana/dashboard_indexing_metrics.json)*
- Query Service *(https://raw.githubusercontent.com/lucidworks/fusion-cloud-native/master/monitoring/grafana/dashboard_query_pipeline.json)*
- Gateway Metrics *(https://raw.githubusercontent.com/lucidworks/fusion-cloud-native/master/monitoring/grafana/dashboard_gateway_metrics.json)*
- JVM Metrics *(https://raw.githubusercontent.com/lucidworks/fusion-cloud-native/master/monitoring/grafana/dashboard_jvm_metrics.json)*
- Core *(https://raw.githubusercontent.com/lucidworks/fusion-cloud-native/master/monitoring/grafana/dashboard_solr_core.json)*
- Node *(https://raw.githubusercontent.com/lucidworks/fusion-cloud-native/master/monitoring/grafana/dashboard_solr_node.json)*
- System *(https://raw.githubusercontent.com/lucidworks/fusion-cloud-native/master/monitoring/grafana/dashboard_solr_system.json)*
- Kube Metrics *(https://raw.githubusercontent.com/lucidworks/fusion-cloud-native/master/monitoring/grafana/kube_metrics.json)*

Banana Dashboards

Dashboards are Fusion's built-in analytics component. Most of the components of dashboards, called *panels*, are for data visualization. Some panels display textual information; for example, records in a table or documents.

	Additional analytics functionality is available in the *DevOps Center* and *App Insights*. A Fusion license is needed to enable App Insights.

Dashboards permit near-real-time views of data. For some types of panels, you can configure how frequently the panel gets new data.

Dashboards also permit users to interact with data. For example, users can submit queries, select facets, and zoom in and out on histograms.

A Fusion dashboard contains one or more controls for search query inputs and one or more quantitive displays over the results for that query.

Dashboards run as a client-side application in a web browser, using JavaScript components for HTML5. Client-side JavaScript provides a dynamic, responsive browsing experience. Solr facets provide the quantifications required for visualizations, for example, charts, graphs, tables, and maps (for geospatial data). Dashboards can also have tabular displays for drilling down to the individual documents in a results set.

The underlying browser application is Banana, an open-source visualization tool based on Kibana 3 *(http://www.elastic.co/guide/en/kibana/3.0/index.html)*. Where Kibana communicates with Elasticsearch, Banana communicates with Solr. Banana can be downloaded from: https://github.com/LucidWorks/banana/ *(https://github.com/LucidWorks/banana/)*.

Fusion dashboards were first developed for log file analysis. Log file entries contain a timestamp plus some amount of additional event information. Fusion's date/time processing aggregates these timestamps at different levels of granularity allowing for visualization of historical trends in system and user behaviors.

See these additional topics for details:

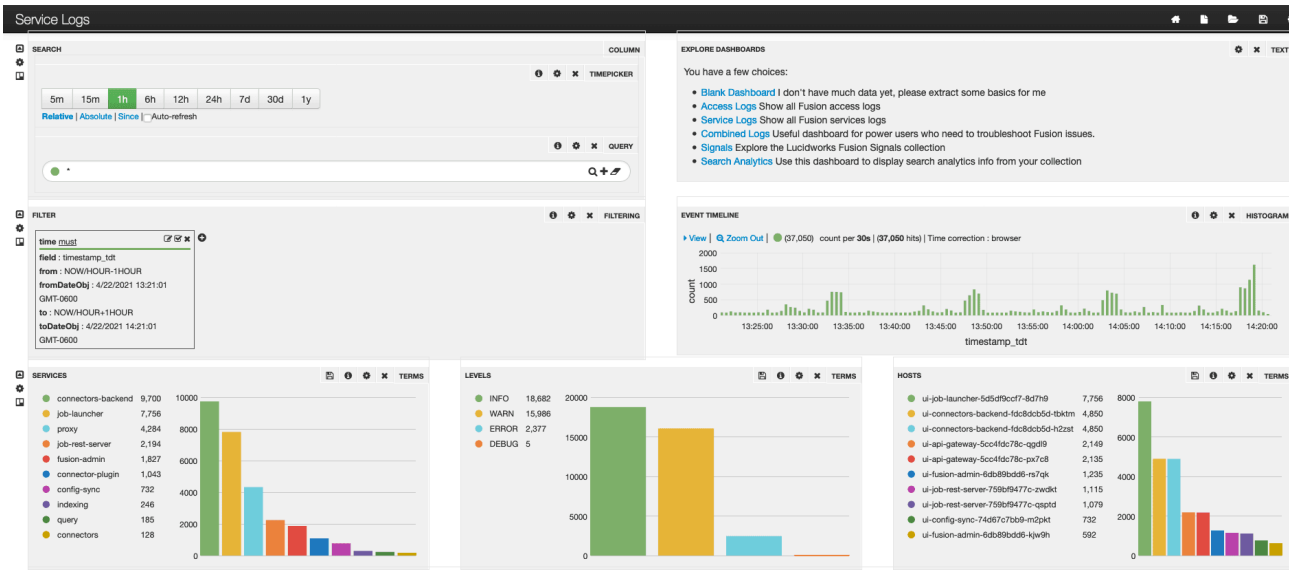
- *Create Dashboards*
- *Use Dashboards*
- *Manage Dashboards*
- *Input Panels*
- *Display Panels*

Built-in Dashboards

Fusion ships with built-in dashboards. To access these dashboards from the Fusion UI, click **Analytics > Dashboards**.

Service Logs dashboard

The Service Logs dashboard (`lucidworks-service-logs.json`) is also the **default** dashboard (`default.json`).



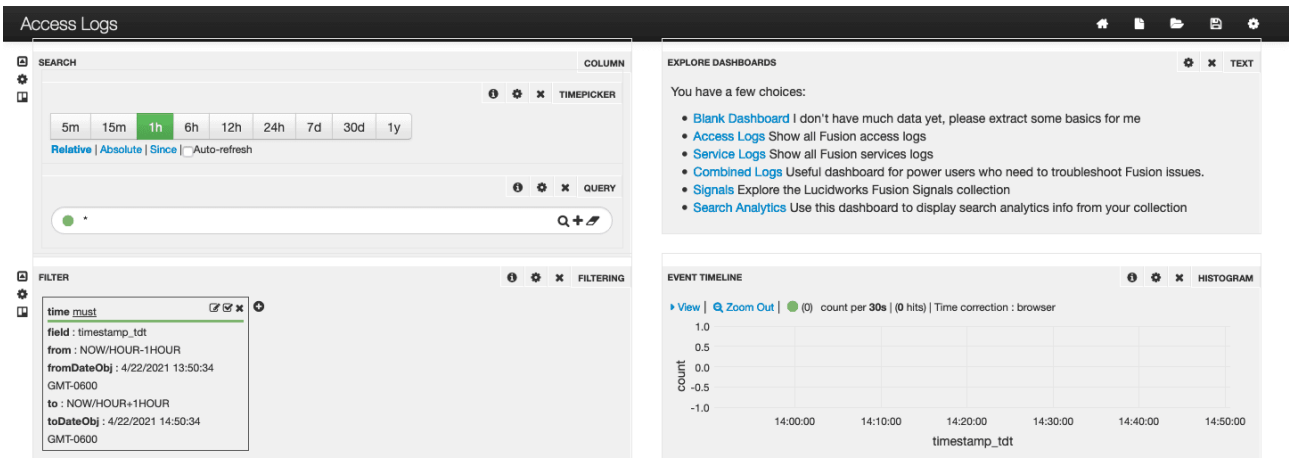
You can modify the default dashboard to suit your needs. To set a different dashboard as the browser default, click the dashboard you want to set as the default, then click **Save > Set as Browser Default**.

To open the default dashboard from the Fusion workspace, click Analytics > **Dashboards** or System > **Log Viewer**.

Alternatively, on any dashboard, click **Goto saved default** .

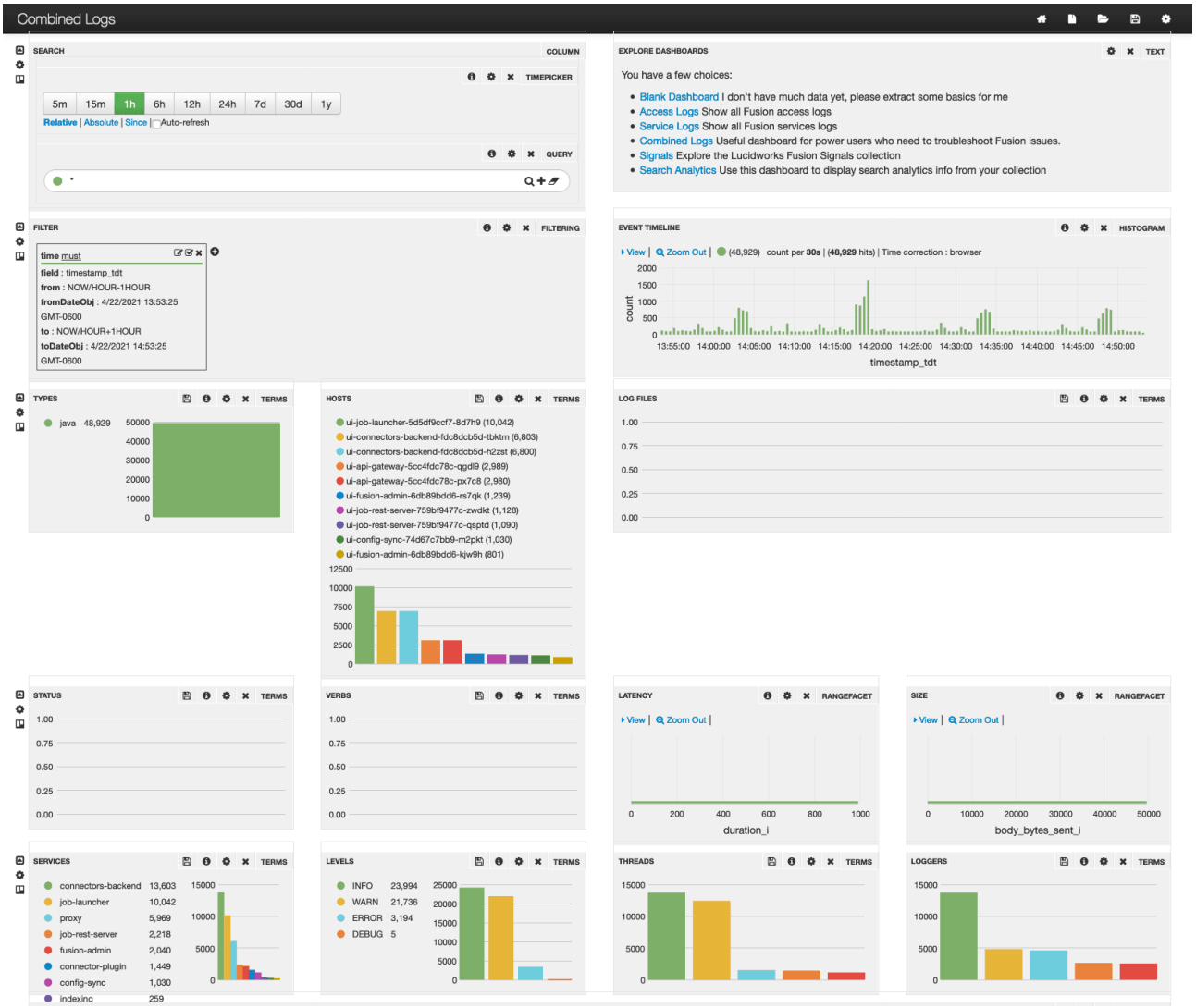
Access Logs dashboard

The Access logs dashboard (`lucidworks-access-logs.json`) lets you analyze logs from the `system_logs` collection where `type_s:http`, that is Fusion access over the Web, for example, from the Fusion UI and web apps that submit Fusion requests.



Combined Logs dashboard

The Combined Logs dashboard (`lucidworks-combined-logs.json`) lets you analyze all logs from the `system_logs` collection.

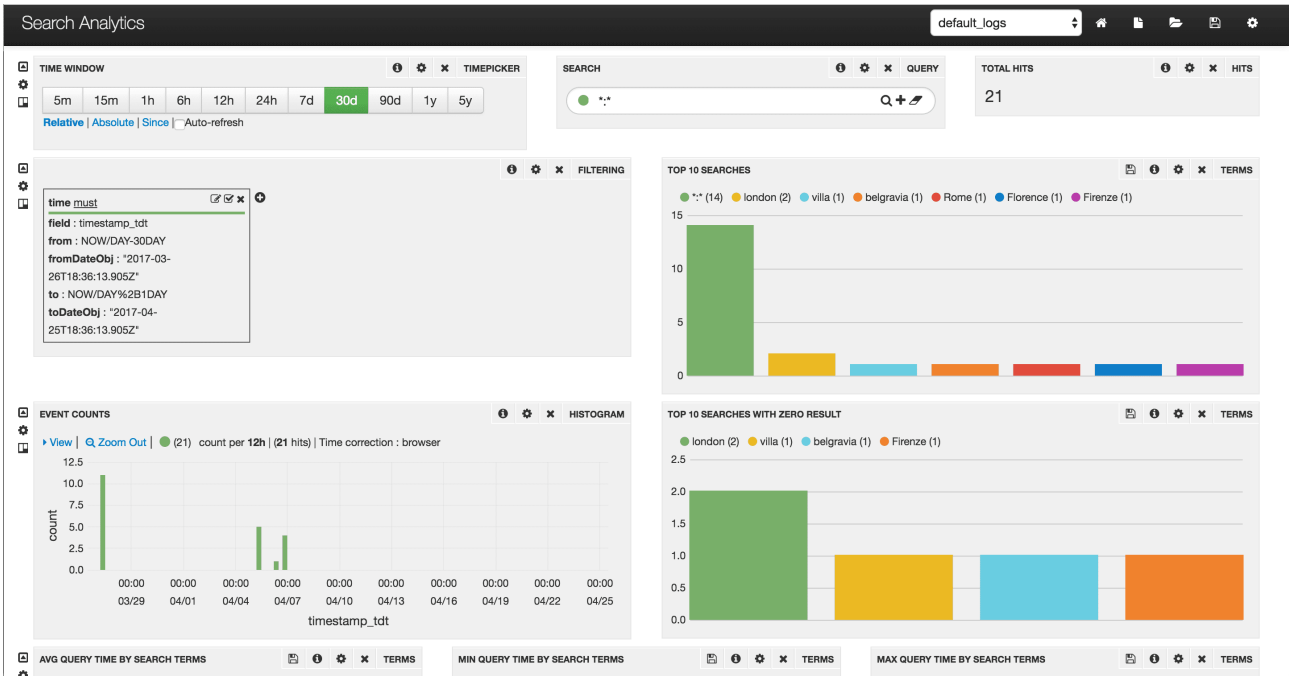


Blank dashboard

An entirely blank dashboard (`blank.json`) is available. To open this dashboard from the default dashboard, click **Blank Dashboard** at the top right.

Search Analytics dashboard

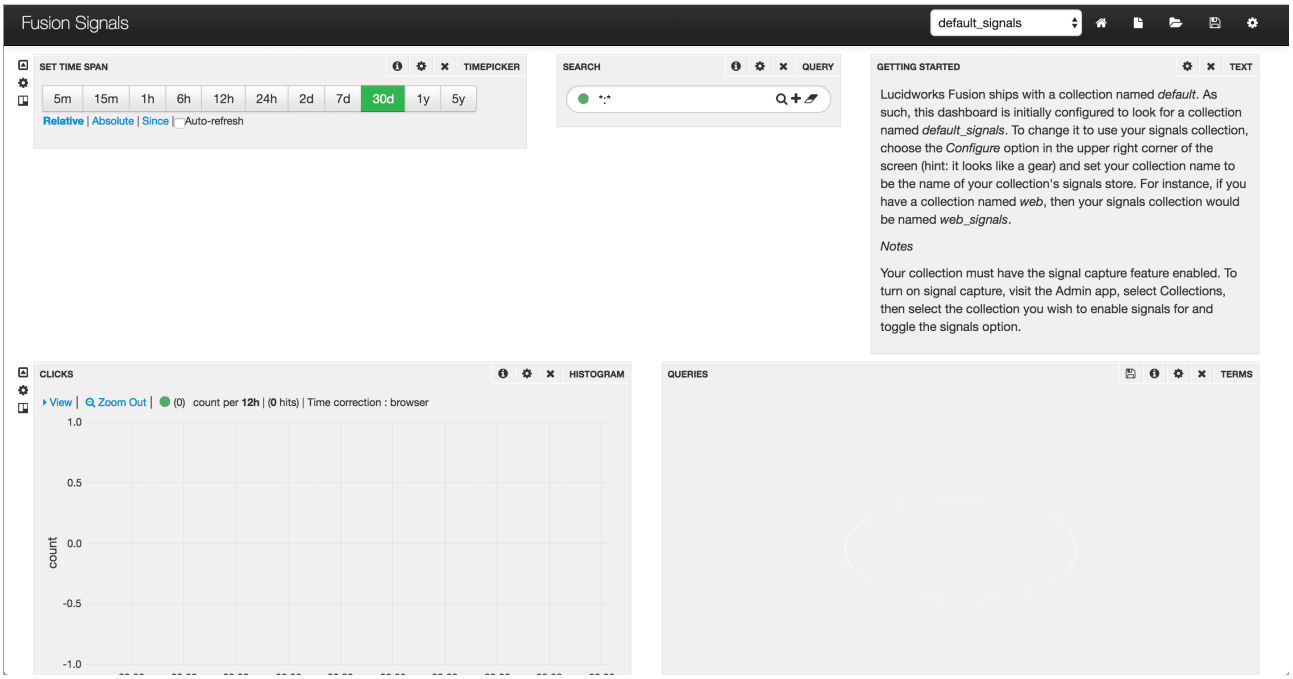
The Fusion Search Analytics dashboard (`lucidworks-searchanalytics.json`) displays search analytics derived from collection logs and from queries to Solr (`/api/solr`).



To open the Search Analytics dashboard from the default dashboard, click **Search Analytics** at the top right.

Fusion Signals dashboard

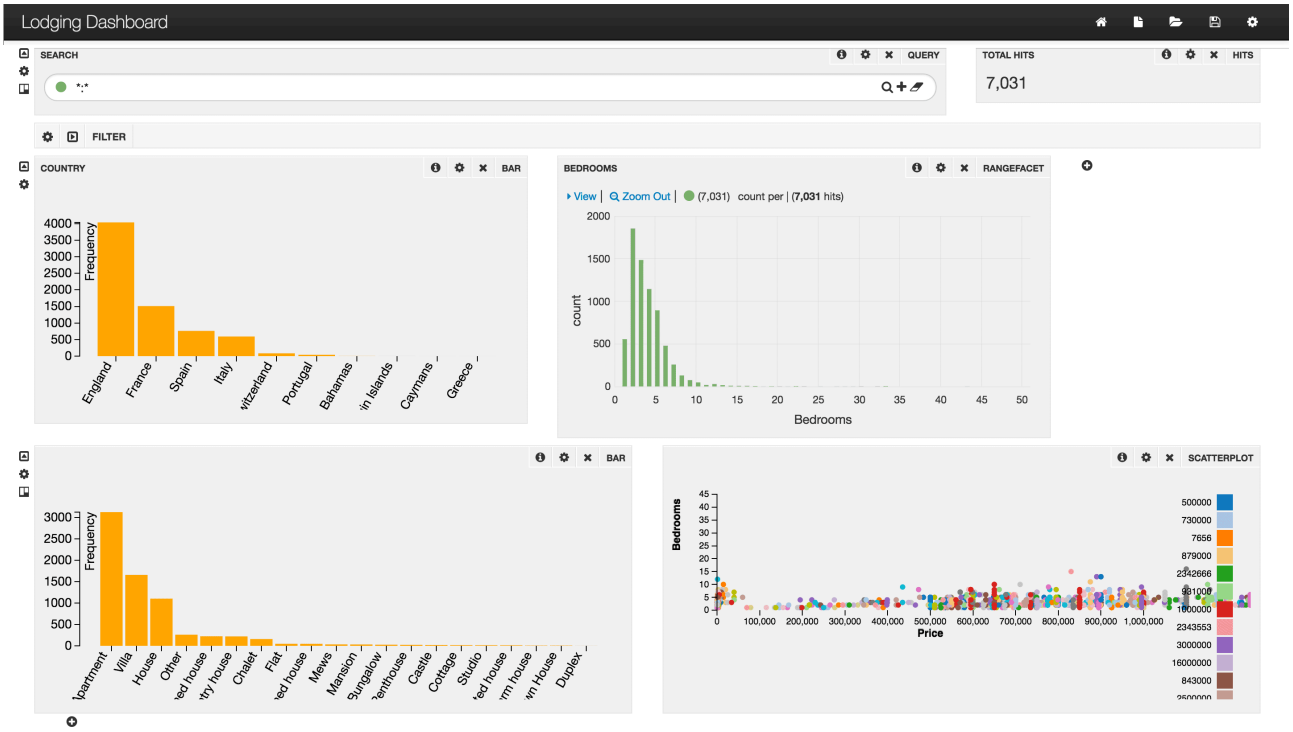
The Fusion Signals dashboard (`lucidworks-signals.json`) is a time-series dashboard that you can use to monitor signals collections. It uses the signal timestamp as the time field.



To open the Fusion Signals dashboard from the default dashboard, click **Lucidworks Fusion Signals** at the top right.

Default Non-Time-Series dashboard

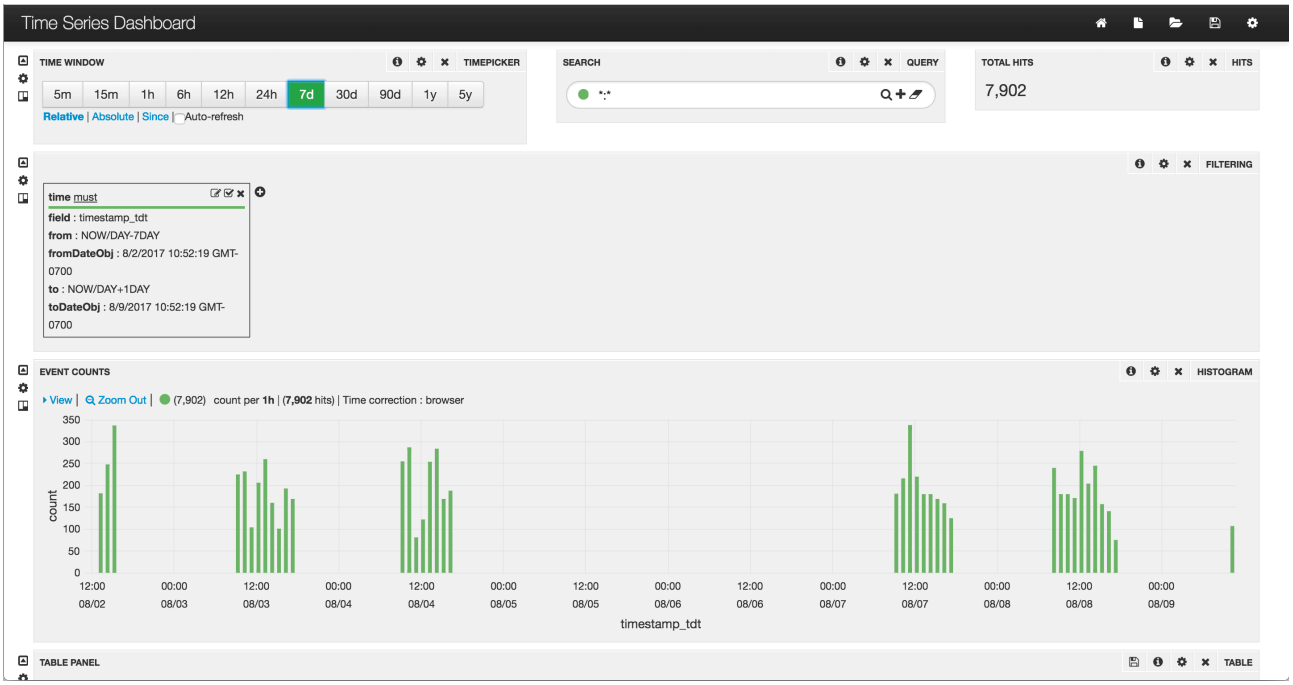
Fusion includes a default non-time-series dashboard (`default-nts.json`). This is an example:



To open the default non-time-series dashboard, at the top right of any dashboard, click **New > Non-time-series dashboard**.

Default Time-Series dashboard

Fusion includes a default time-series dashboard (`default-ts.json`). This is an example:



To open the default time-series dashboard, at the top right of any dashboard, click **New > Time-series dashboard**.

Display Panels

Display panels on a dashboard display information about the data in a single collection. Fusion has these types of display panels:

- **Layout.** Use these display panels to organize the panels on a dashboard.
- **Textual Information.** Use these panels to display textual information. For example, use a Table panel to displays the values of fields in a set of records.
- **Graphical Visualization.** Use these panels to help users to visualize data. For example, display category data in a bar chart and geographical data on a map.

What Data is Displayed

These things determine which data Fusion displays in a display panel:

- Where the query is sent (to a Fusion query pipeline or to Solr)
- The collection that contains the data. You can specify this when you create a dashboard, or permit the user to choose the collection.
- Global query parameters (optional)
- Input panel configuration settings (possibly including a panel query)
- A user's selection of the collection (if permitted)
- A user's query
- A user's interactions with the data

Layout Panels

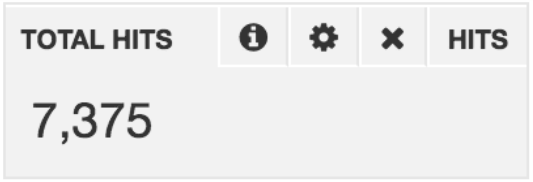
These layout panels can help you organize a dashboard:

- **Column.** Lay out panels in a column within a row or part of a row. (Instead of the usual left-to-right layout within rows.)
- **Text.** Add text to a dashboard, for example, to instruct the user regarding how to use the dashboard, or to describe the data.

Textual Information Panels

These panels display textual information:

- **Doc Viewer.** Display documents as single pages of information, and let the user page through documents. Paging through documents does not affect the data displayed in other panels.
- **Full Text Search.** Provide full text search capability.
  Note: Make a Full Text Search panel wide. Start with Span = 12. Depending on the content, you might be able to make it narrower. If you make a Full Text Search panel too narrow, content and controls can be inaccessible.
- **Hits.** A Hits panel displays statistical information about the hits returned by the query, including all filters that are applied to the query. The default information is the count (number of hits). You can display the count, minimum, mean, maximum, sum, standard deviation, the sum of squares, and/or the number of hits that lack a value for a field.

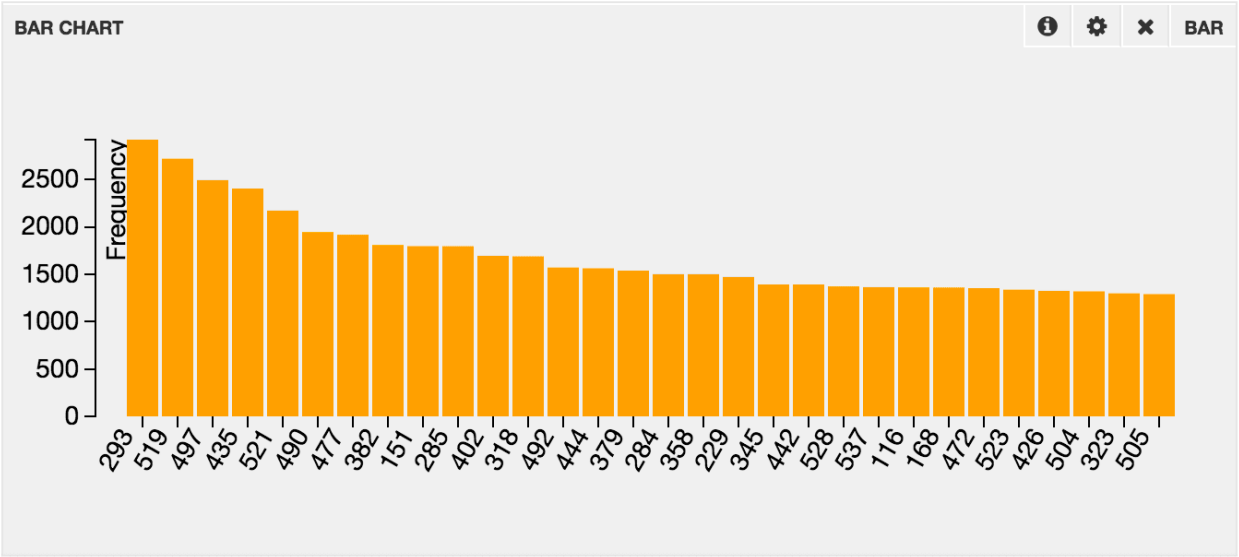


- **Table.** Display data in a table.
  Note: Make a Table panel wide. Start with Span = 12. Depending on the content, you might be able to make it narrower. If you make a Table panel too narrow, content and controls can be inaccessible.

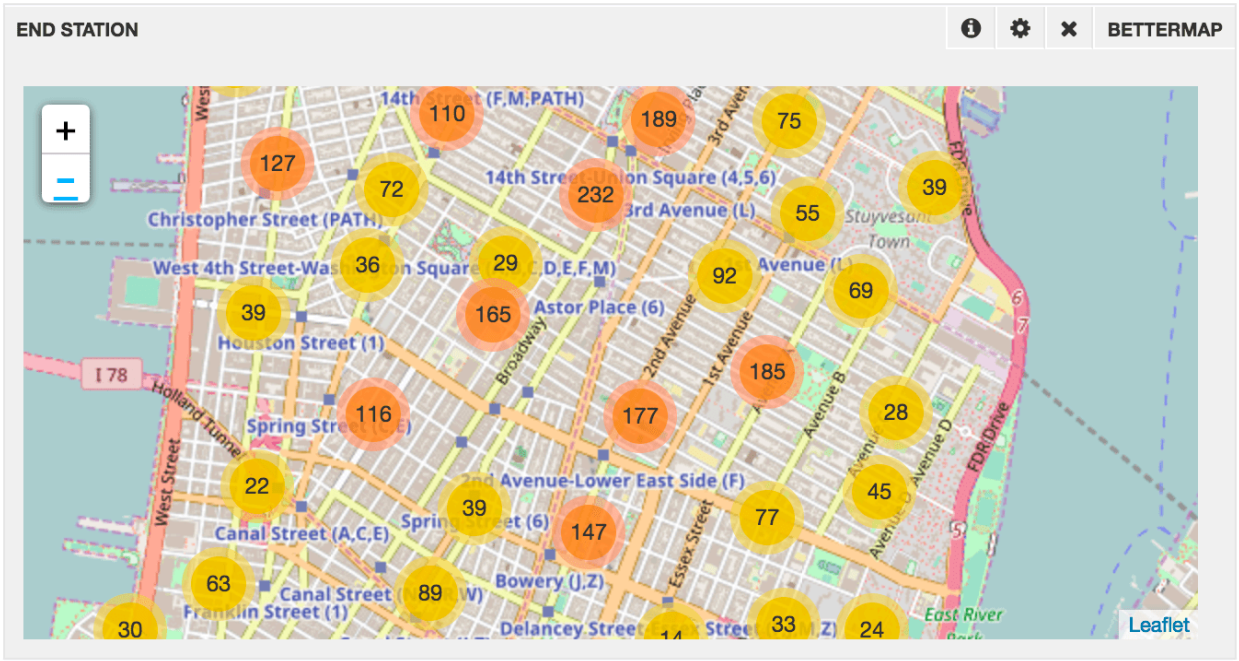
Graphical Visualization Panels

Most display panels are graphical visualization panels. They visually present the data requested in the input panel(s), subject to configuration of the display panel and to any global filtering. Panels for visualizing information graphically are:

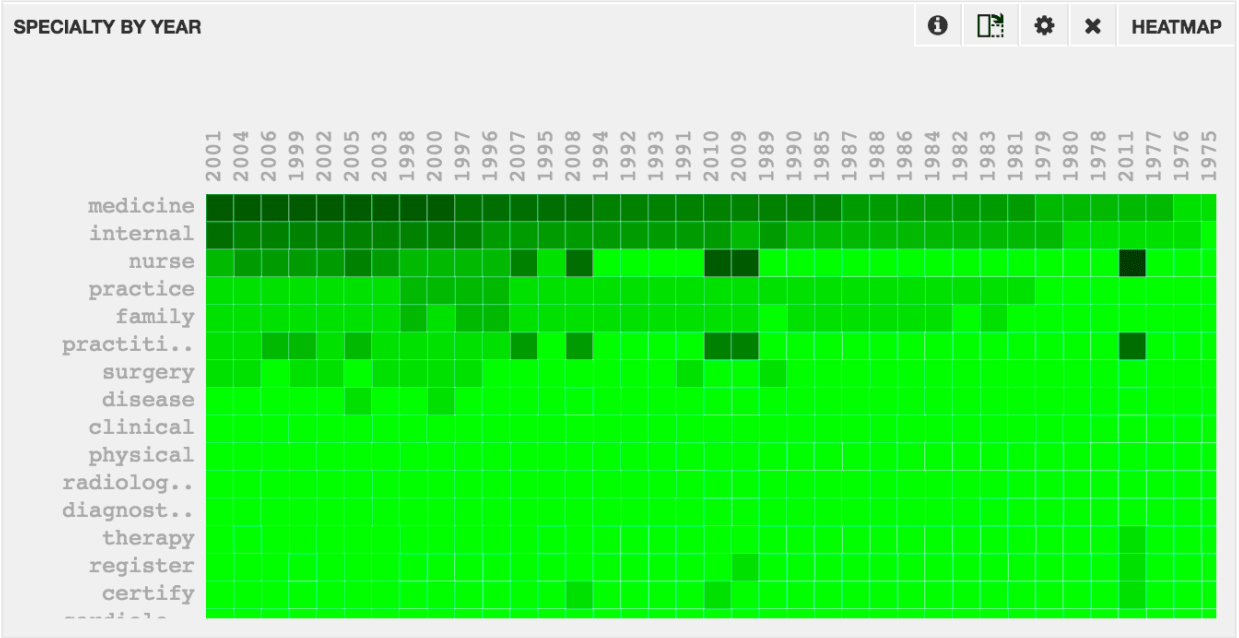
- **Bar.** Graph data as frequencies in a bar chart.



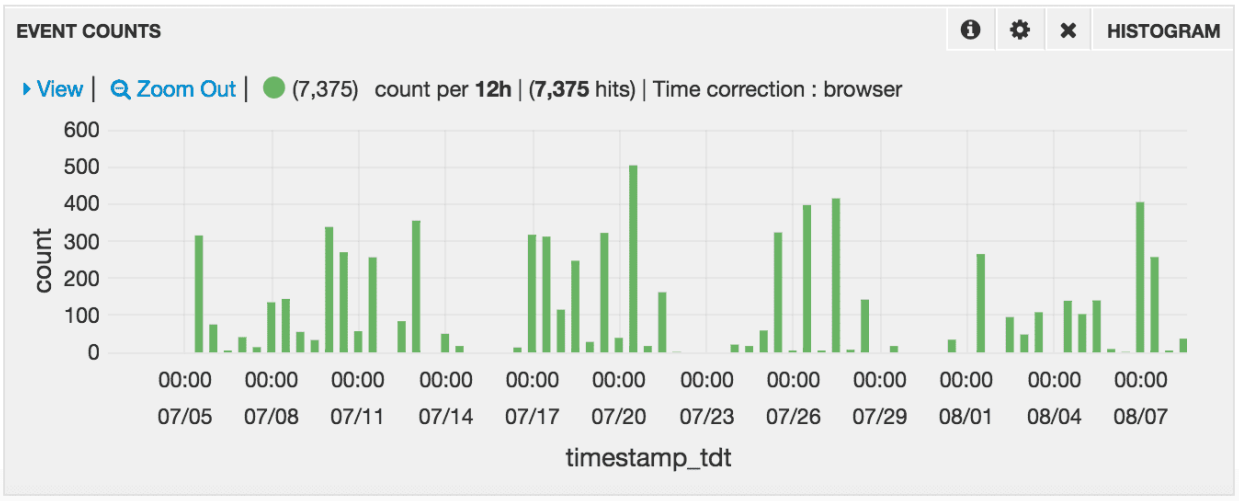
- **Better Map.** Display geolocated points in clustered groups on a map. As you zoom out, points are clustered into fewer groups. As you zoom in, points are clustered into fewer groups (and at some point are not clustered). The Better Map panel *does not* use the geospatial search capabilities of Solr. It transfers more data than the Map panel and generally requires more computation, while showing less data. If you have a time filter, the panel will show the most recent points in your search, up to your defined limit. This panel is best used after filtering the results through other queries and filter queries, or when you want to inspect a recent sample of points.



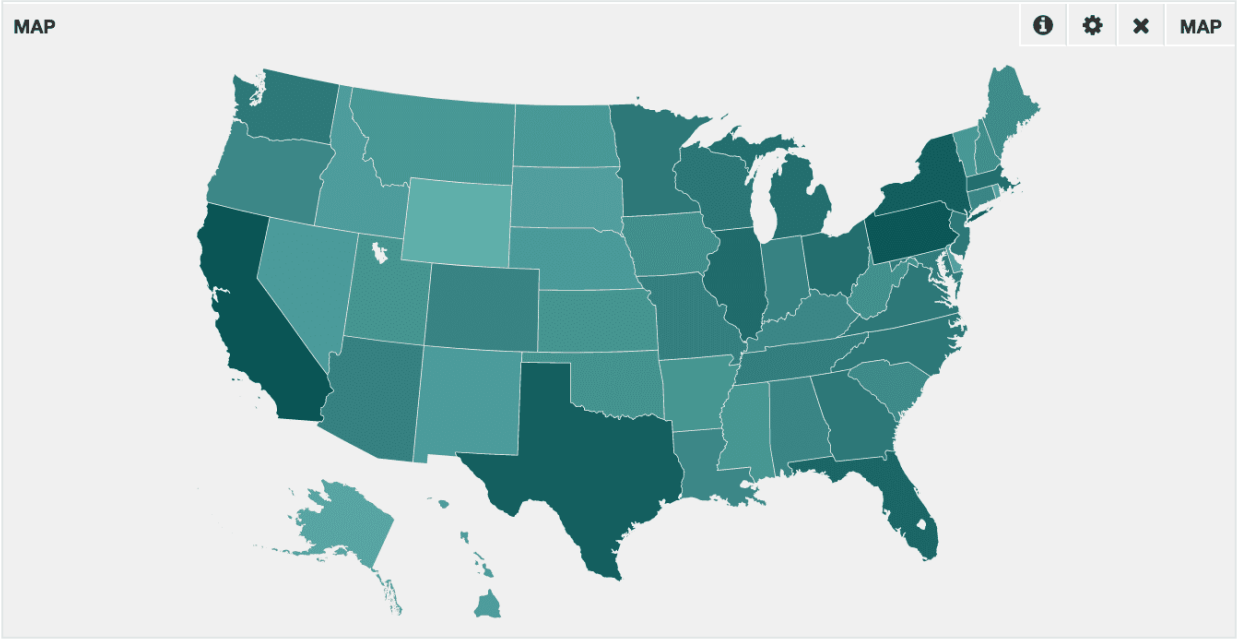
- **Heat Map.** Display a heat map, that is, a graphical representation of data along two facet axes.



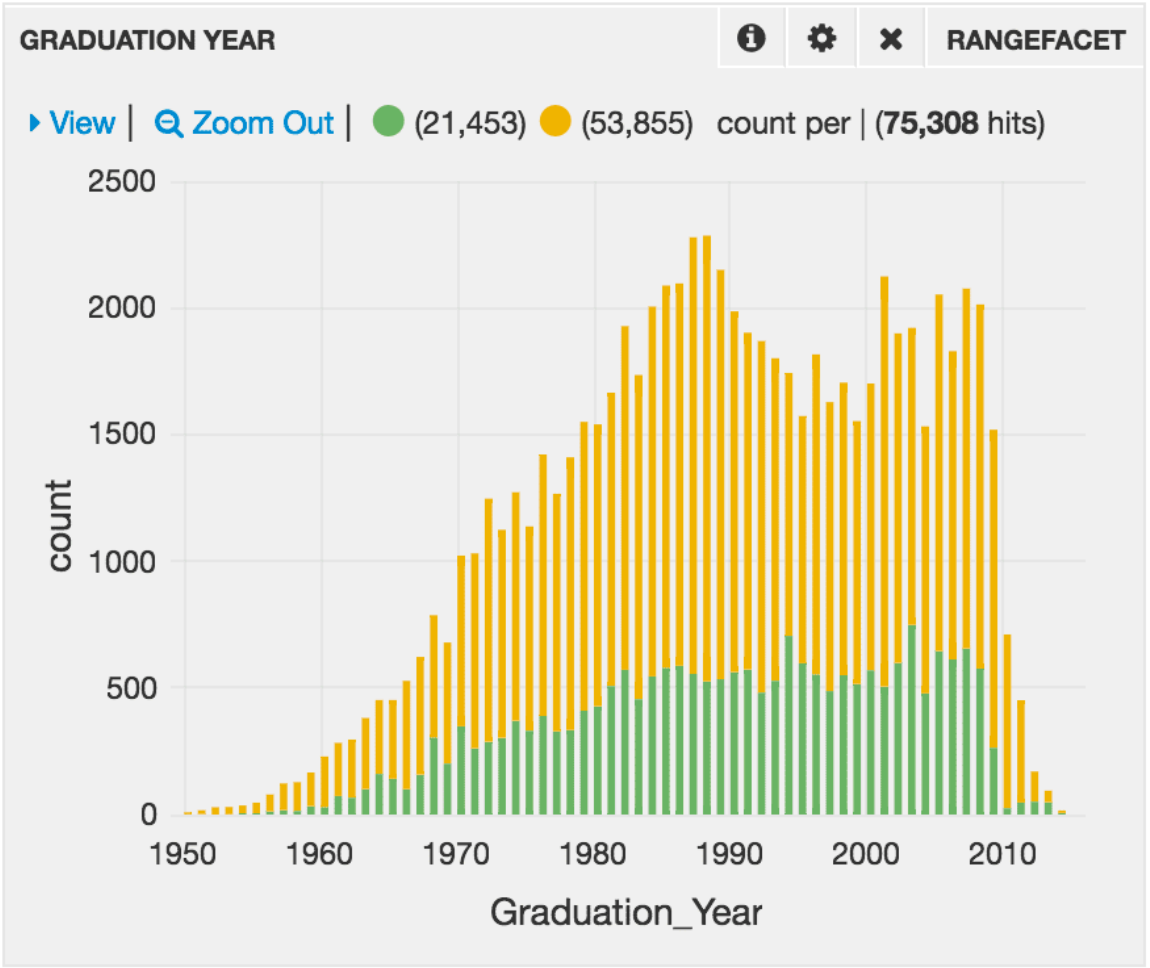
- **Histogram.** Display a histogram. When used in `count` mode, this is a bucketed chart of the current query, including all applied time and non-time filters. When used in `values` mode, the histogram plots the value of a specific field over time, and lets the user group the values based on the values of a second field.



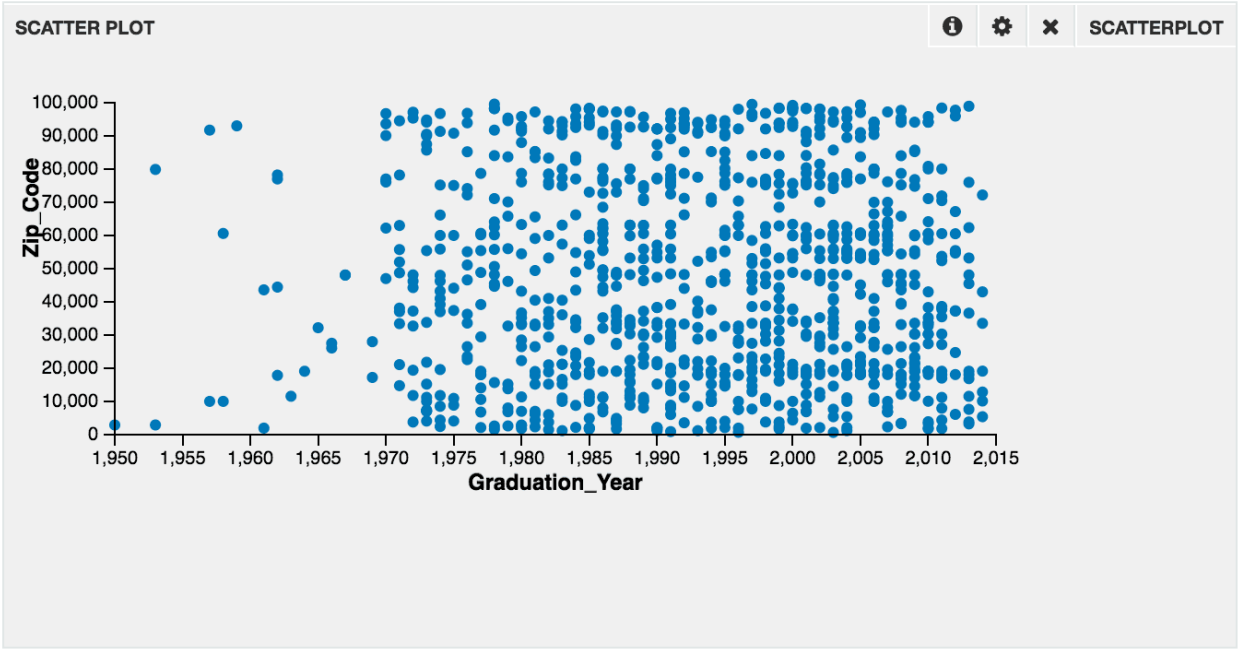
- **Map.** Display a map of shaded regions using a field that contains a 2-letter country code or U.S. state code. Regions with more hits are shaded darker. Instead of a count, you can choose to shade regions based on the minimum, maximum, mean, or sum. The map panel uses facets, so it is important that you set field values to the appropriate 2-letter codes at index time.



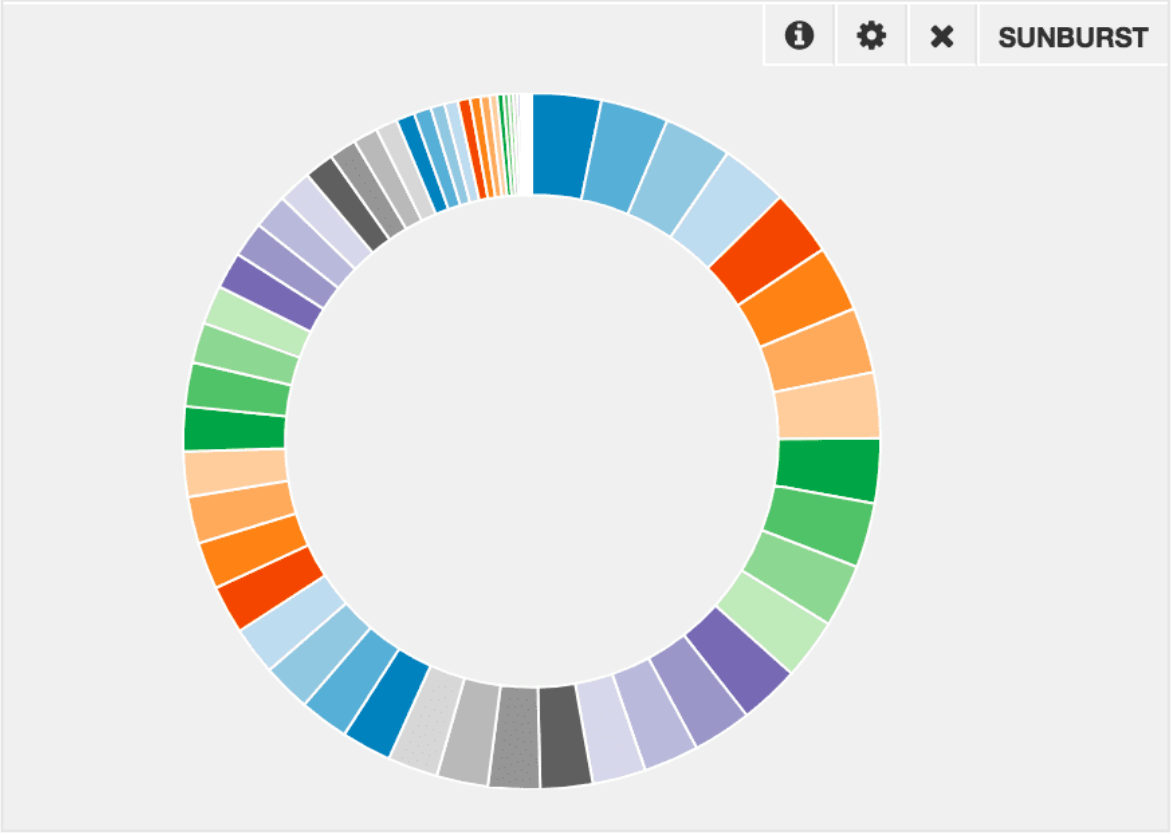
- **Multi-series.** Graph multiple fields of the same type that vary with time in the same graph. The X-axis is used for the time series field. Values in the other fields are graphed along the Y-axis.
- **Range Facet.** Display a histogram of a numeric field. The Range Facet panel is similar to the time series histogram. It lets you select ranges and zooming in/out to the desired numeric range. Range selections in the panel are reflected across the entire dashboard. With multiple search boxes in a Query panel, the Rangefacet panel can display multiple data sets, as shown in this example:



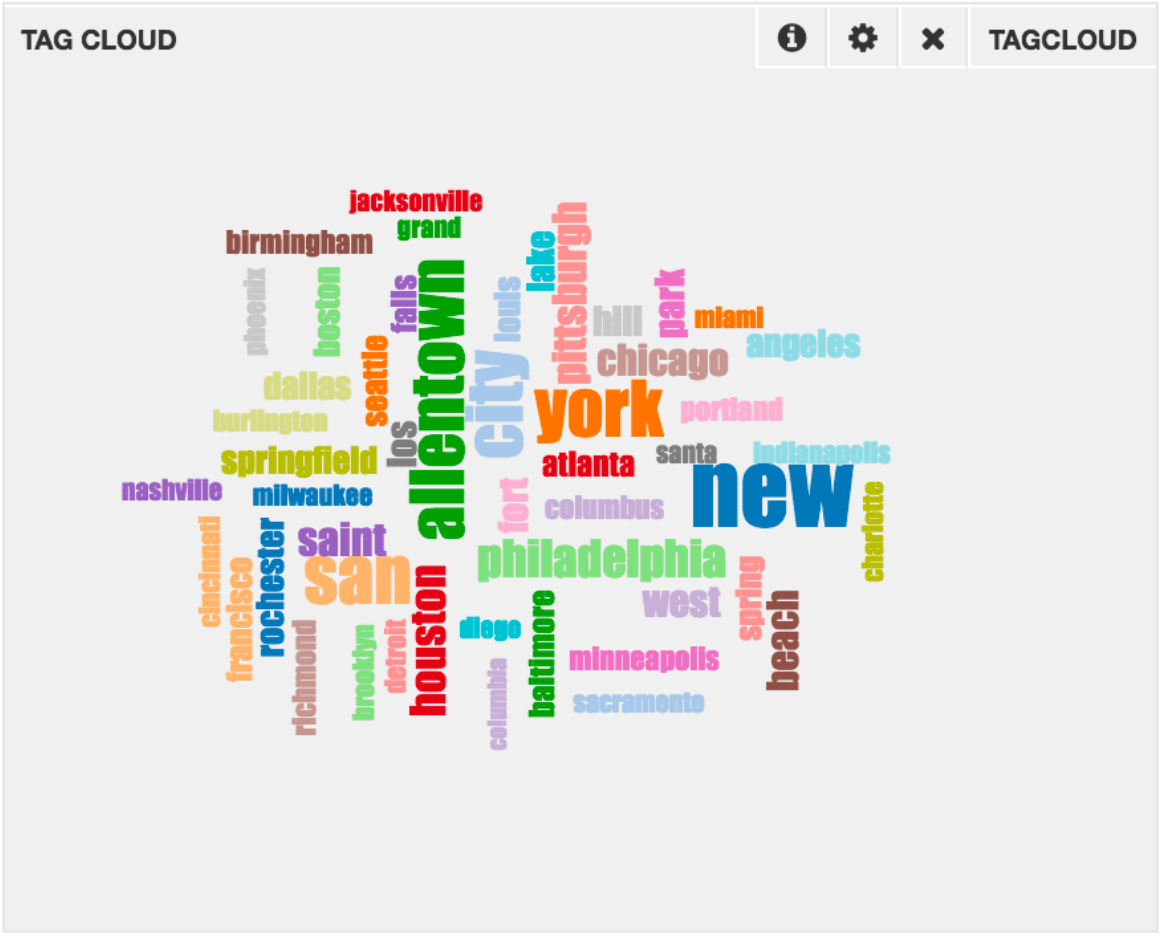
- **Scatter Plot.** Display a scatter plot between two variables or four variables.



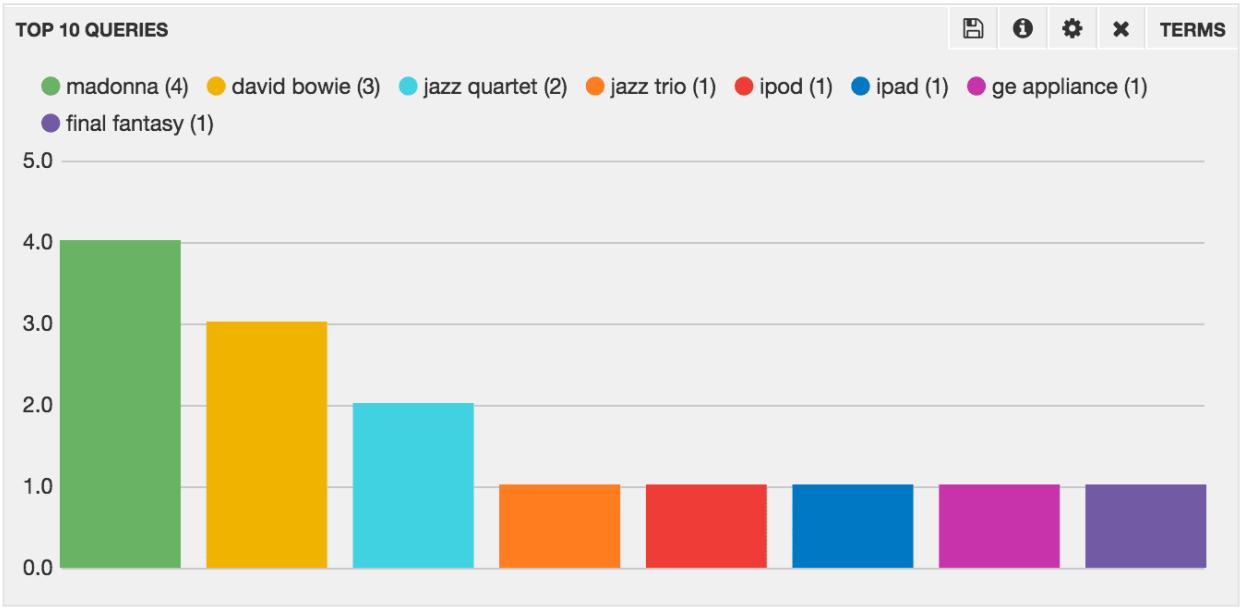
- **Sunburst.** Display a sunburst plot based on facets.

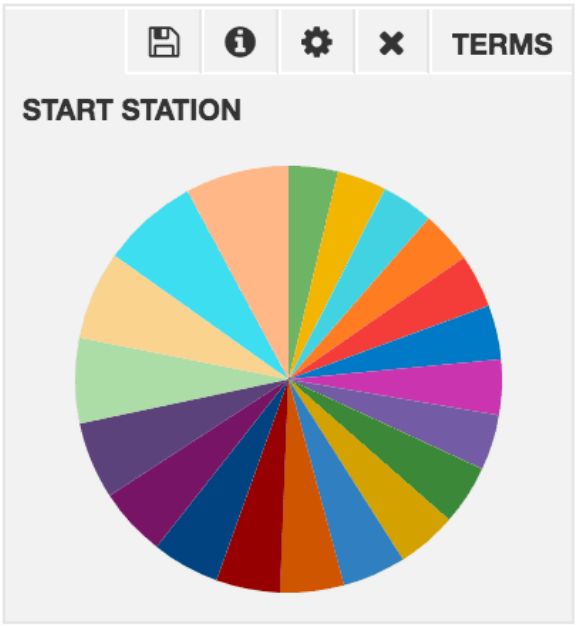


- **Tag Cloud.** Display a tag cloud of the top N words in a field.



- **Terms.** Display the count, minimum, maximum, mean, or sum for a Solr facet as a bar chart, pie chart, or table.



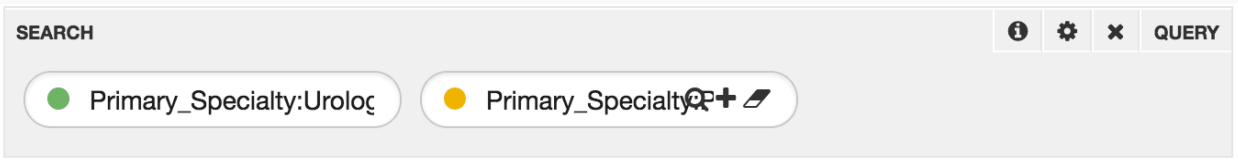

TERMS
START STATION

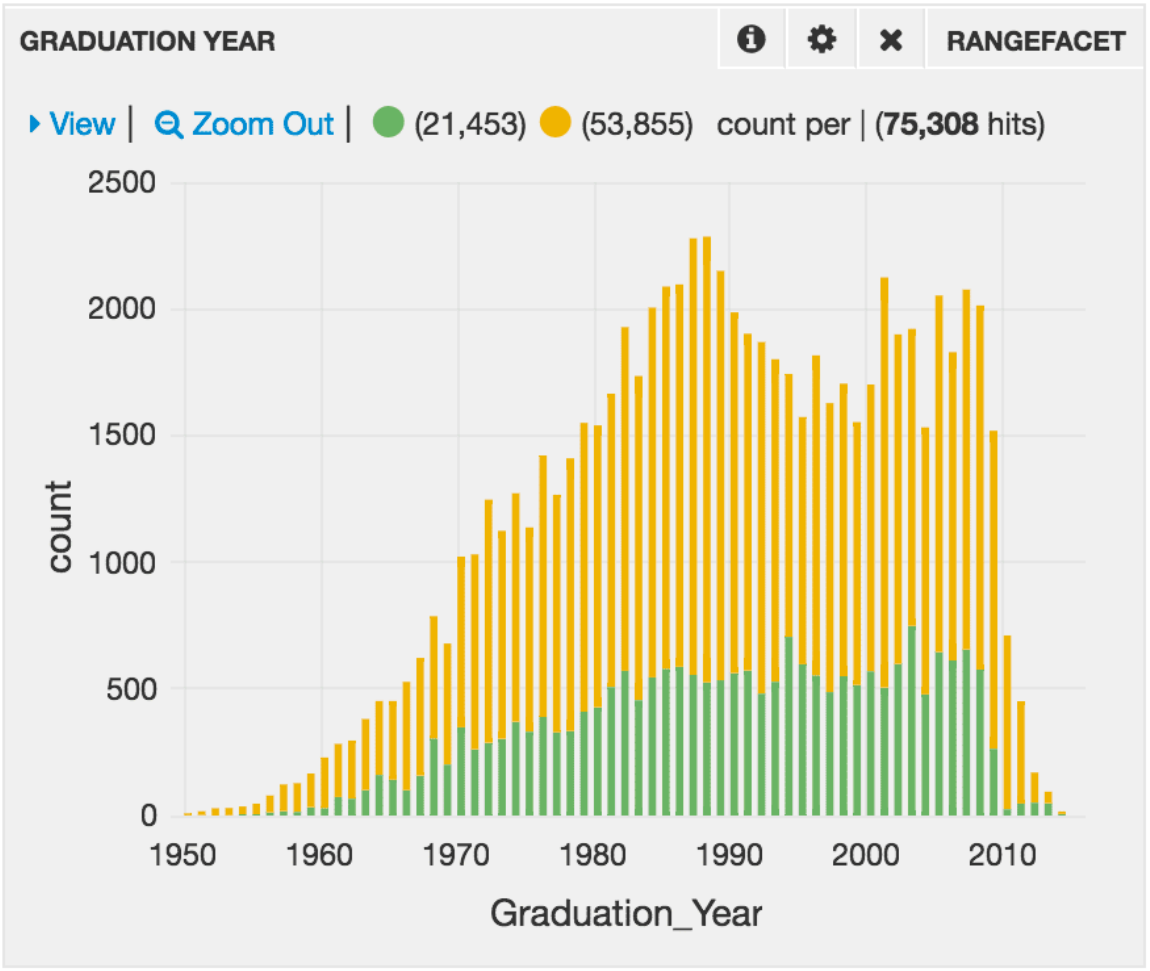
Input Panels

Input panels let you control which data output panels display. The values you specify in input panels become parts of the Solr queries that output panels use to obtain data.

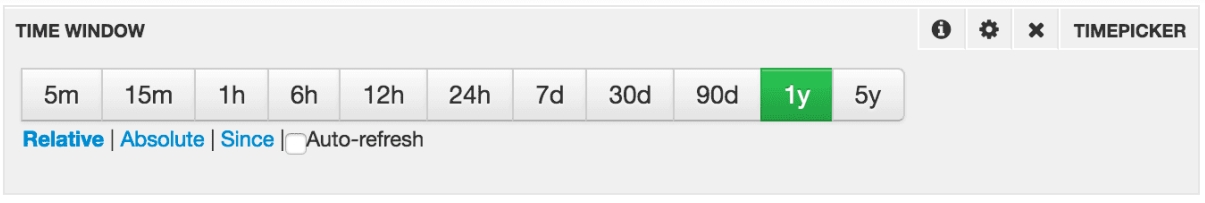
You can use these input panels:

- **Query Panel.** Enter a free-form query (one or more query terms) in a search bar. Add additional search bars to a query panel to submit separate queries. Some visualization panels (for example, Rangefacet) keep the data separate so you can compare it. This is an example of a Query panel with two search boxes. The parameters for query strings are `Primary_Specialty:Urology` and `Primary_Specialty:Psychiatry`. Here is the Query Panel and the resulting Rangefacet panel:



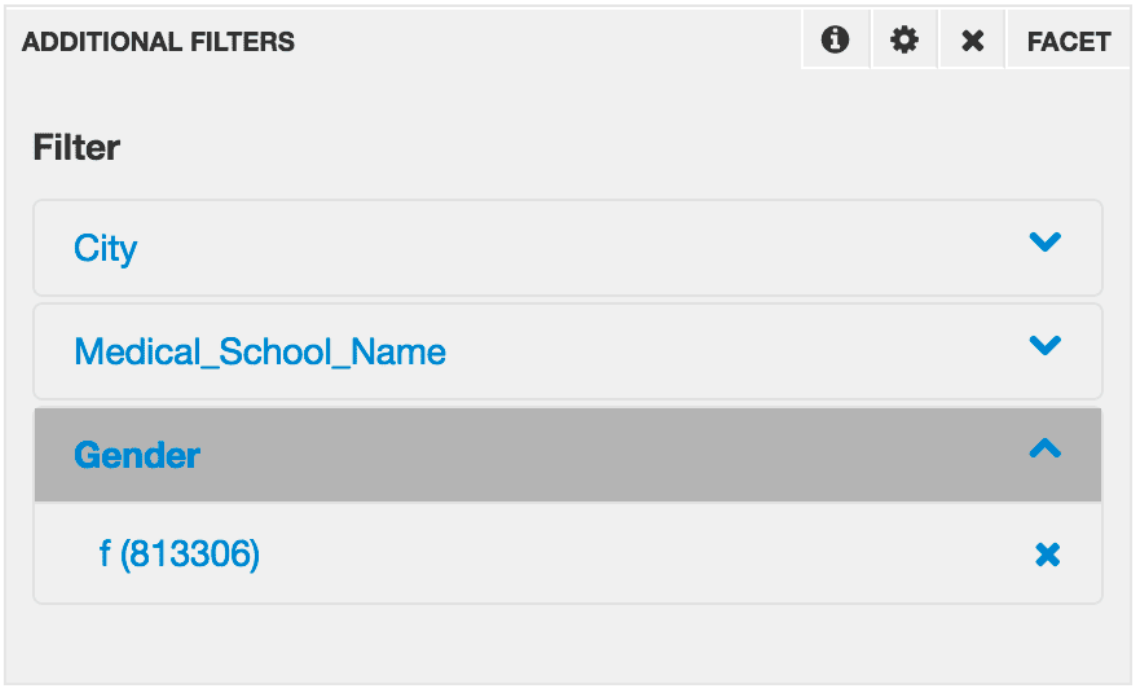


- **Time Picker Panel.** Apply a time range to time-series data. The time range can be:
  - Relative.* A time range starting now and reaching backward in time, for example, the last 15 minutes or one hour

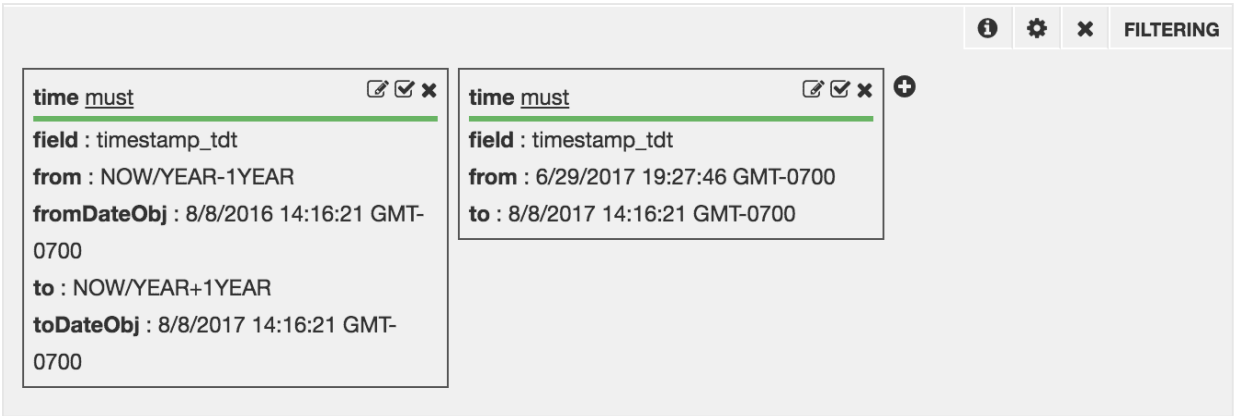


  - Absolute.* A specific time-and-date range, for example, from 06/01/2017 00:00:00 to 06/30/17 23:59:59

  - Since.* A time range since a specific date and time, for example, since 01/01/2017 00:00:00.
- **Facet.** A Facet panel can facet any data field.



- **Filtering.** A Filtering panel lets you apply field-based filters to the hits returned by the query. The filters apply to all display panels.



	You must use a Filtering panel somewhere on your dashboard, so that all panels work correctly when you interact with data.

Query Syntax

Enter a search term or phrase in the search box of the Query panel. Autocompletion provides a list of possibly related prior searches. Finish typing your search query or select an autocompleted query, and then click **Search**.

Enter queries in a Query panel using Apache Lucene Query Parser syntax *(https://lucene.apache.org/solr/guide/the-standard-query-parser.html#the-standard-query-parser)*. You enter the parameter for the query string (for example, `Susan` or `Gender:F`), not the query string (for example, `q=Susan` or `q=Gender:F`).

Rules for the Simplest Cases

Here are some syntax rules for the simplest cases:

- A single term is either `field:value` or `value`. With `value`, the search is over all fields.

- For an exact-case match, you must specify the field name.
- Surround a term that contains spaces in *double quotation marks* (" ").
- You cannot specify `*:value` or `*:"value"` to search over all fields; that syntax does not work.
- To retrieve all records, use the search term `*:*`.
- For OR logic, enter `OR` between the terms, or just use spaces. For AND logic, enter `AND` between the terms.

Examples

These are examples of the query syntax:

Goal	Syntax and example
Single term in any field; no blanks in term	`term` (matches any case)
Single term in any field; blanks in term	`"term"` (matches any case)
Multiple terms, each in any field; with OR logic; no blanks in terms	`term1` `term2` …
Multiple terms, each in any field; with OR logic; blanks in terms	`"term1"` `"term2"` … (matches any case)
Multiple terms, each in any field; with AND logic (in the same record); no blanks in terms	`term1 AND term2` …
Multiple terms, each in any field; with AND logic (in the same record); blanks in terms	`"term1" AND "term2"` … (matches any case)
Single term in a specific field; no blanks in term	`field:term` (matches any case)
Single term in a specific field; blanks in term	`"field:term"` (matches *exact* case)
Multiple terms, each in a specific field; with OR logic; no blanks in terms	`field1:term1` `field2:term2` …
Multiple terms, each in a specific field; with OR logic; blanks in terms	`"field1:term1"` `"field2:term2"` … (matches *exact* case)
Multiple terms, each in each in a specific field; with AND logic (in the same record); no blanks in terms	`field1:term1 AND field2:term2` …
Multiple terms, each in a specific field; with AND logic (in the same record); blanks in terms	`"field1:term1" AND "field2:term2"` … (matches *exact* case)

For more information about the query syntax, see Standard Query Parser Parameters *(https://lucene.apache.org/solr/guide/the-standard-query-parser.html#the-standard-query-parser)*.

	You can use a Text panel to advise the user regarding the syntax of search terms. Also, if you want the dashboard user to explore subsets of the data, use a Filtering panel or a Facet panel to achieve that. do not expect users to enter complex search expressions. For example, add the field `gender` as a facet, instead of expecting the user to add `AND gender:male` to a search expression.

Inspect a Panel Query

You cannot inspect the panel query in the Query panel. You can inspect the panel query in other panels, for example, in a Histogram or Heatmap panel. You can see the contributions that the different parts of the query make. In this example, there are no global query parameters.

Query:

```
start_station_name:"Broadway and E 14 St" AND gender:Male
```

Panel query for a Histogram panel:

The part of the query from the Query panel is the first part, from `q=` through `Male`.

```
q=start_station_name%3A%22Broadway%20and%20E%2014%20St%22%20AND%20gender%3AMale&wt=
json&rows=0&fq=start_time:[2014-01-28T20:16:59.000Z%20TO%202014-03-
15T05:36:55.000Z]&facet=true&facet.range=start_time&facet.range.start=2014-01-
28T20:16:59.000Z&facet.range.end=2014-03-15T05:36:55.000Z&facet.range.gap=%2B12HOUR
```

Prometheus and Grafana in Fusion

Overview

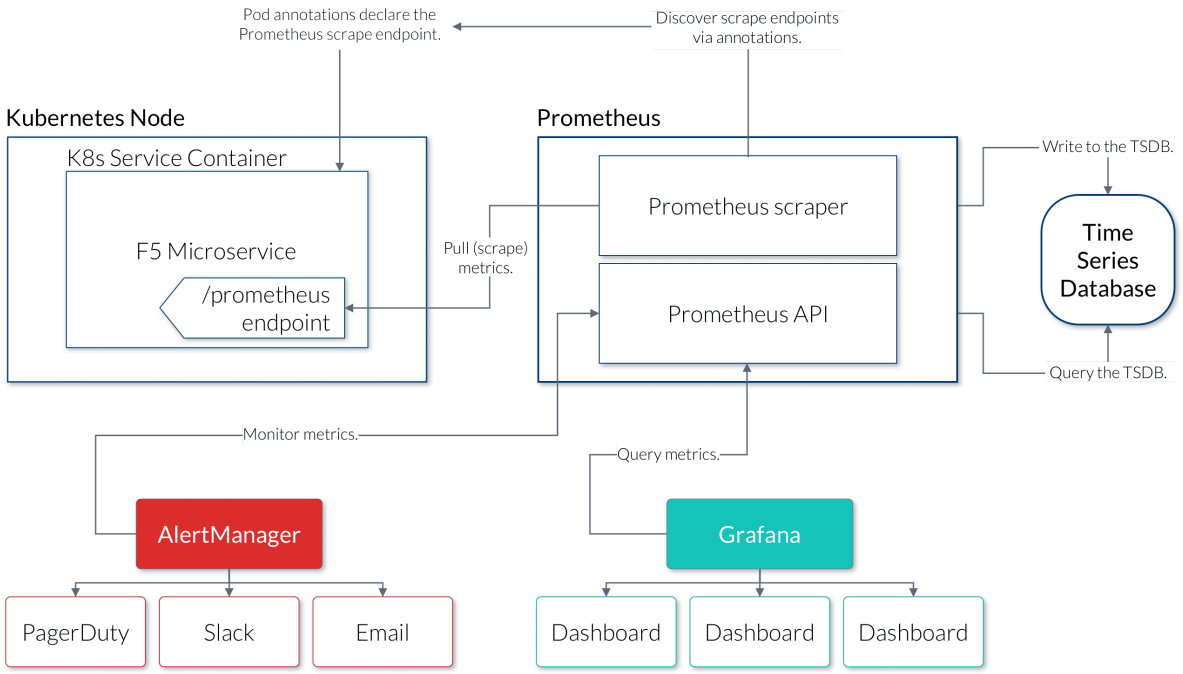
Starting in Fusion 5.0.2, support is added for Prometheus and Grafana for enhanced metrics collection and querying.

Prometheus
Prometheus *(https://prometheus.io)* provides a powerful solution for recording metrics, persisting them to a pluggable time series datastore, and querying them for both visualizations and alerting purposes. Metrics data is pulled from Fusion and stored in a time series datastore.

Grafana
Grafana *(https://grafana.com/grafana)* provides an industry-standard metric visualization tool that supports complex visualizations over any number of different pluggable data sources, organized into panels and managed dashboards. Grafana fully supports Prometheus as a datasource, allowing you to query against it when creating or editing panels in your dashboards.

See *Grafana Dashboards* for information on available Fusion 5 dashboards.

Architecture



Setup and Configuration

See *Install Grafana Dashboards*.

The DevOps Center

Fusion provides support for *Prometheus and Grafana in Fusion* for metrics collection and querying.
However, the DevOps Center is a tool that provides some functions to monitor, troubleshoot, and investigate incidents. It consists of an interactive log viewer, providing views into this Fusion cluster's hosts and services using metrics and events.

System requirements

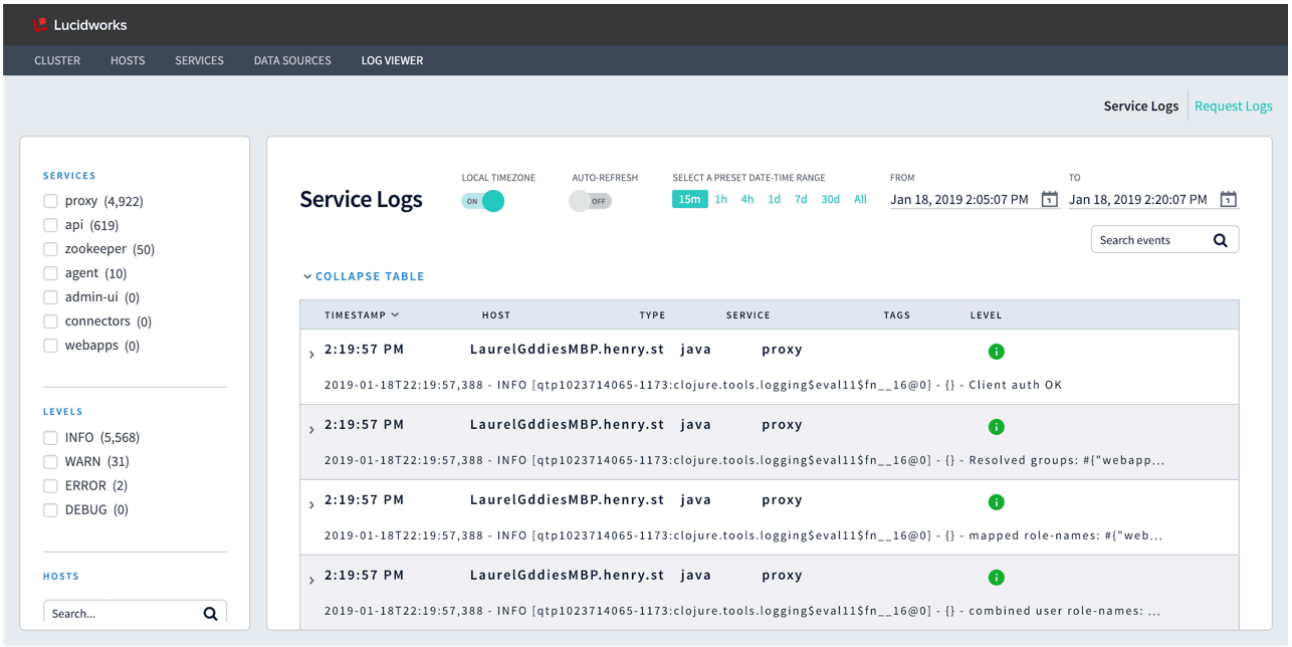
The DevOps Center requires the following:

- Your Web browser must support HTML5.
- Every node that you want to monitor must be running Fusion version 5.x.x.
- Every node that you want to monitor must be running the `agent` and `log-shipper` services.

The DevOps Center is enabled by Fusion's default configuration.

The Log Viewer

The Log Viewer displays service logs and request logs from Fusion's `system_logs` collection.



By default, timestamps are displayed in local time, as determined by your browser. To view timestamps in UTC (unless `default.timezone` is set to another time zone in `fusion.cors`), set **LOCAL TIMEZONE** to "Off".

Auto-Refesh is off by default, to display static log data. To display real-time logs, set this to "On".

See *Export Data from the DevOps Center* to learn how to export log data.

Service logs

Service log files are written to the filesystem by each running service, such as `var/log/api/api.log`, `var/log/proxy/proxy.log`, and so on. Fusion indexes them in the `system_logs` collection with `type=java`.

In the DevOps Center, you can filter service logs by:

- Service
- Log level
- Host

Request logs

HTTP request log files are written to the filesystem by Jetty, at `var/log/proxy/jetty_request_<date>.log`. Fusion indexes them in the `system_logs` collection with `type=http`.

In the DevOps Center, you can filter request logs by:

- HTTP status code
- HTTP method
- Host

Audit logs

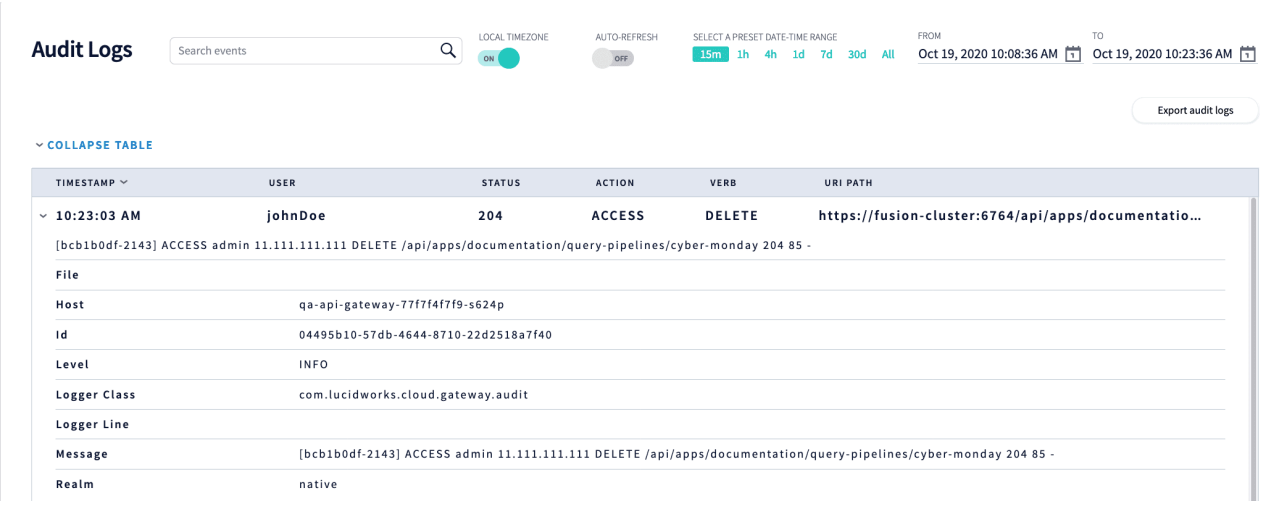
Audit logs provide you with a resource for tracking actions within Fusion, including the date, time, user responsible, and more. Example use cases include determining:

- When a query profile's configuration changed to use `bq` instead of `boost`
- Who created a datasource configuration and why they did so in a particular way
- Whether a deployment is being tampered with

In the DevOps Center, you can filter audit logs by:

- **Status**
- **Action**
- **Verb**

For example, you can examine the audit logs to learn who deleted a query pipeline by faceting by the approximate date range and selecting the **Verb** facet DELETE.



Click the **Export audit logs** for detailed log data, including:

Field	Data
mdc_action_s	`ACCESS`
mdc_request_method_s	`DELETE`
mdc_request_uri_s	`https://FUSION_HOST:FUSION_PORT/api/apps/documentation/query-pipelines/cyber-monday`
mdc_user_agent_s	`Mozilla/5.0 (Macintosh; Intel Mac OS X 10.15; rv:81.0) Gecko/20100101 Firefox/81.0`
mdc_username_s	`johnDoe`
timestamp_tdt	`["2020-10-19T16:23:03.041Z"]`

Fusion 5 Frequently Asked Questions

Does Fusion use Custom Resource Definitions?

Yes, Fusion 5 does utilize custom resource definitions. These CRDs are part of the install and are critical to the application. All CRDs are namespaced, following Kubernetes best practices for utilizing CRDs in a multi-tenant environment.

Are any services installed at the cluster level?

Yes, two services, Argo and Seldon-core are installed at the cluster level. These services, which are part of the ML/AI service infrastructure for Fusion, use web hooks, which must be installed at the cluster level. These web hooks are namespaced, following best practices for utilizing web hooks in a multi-tenant environment.

Can I install Fusion 5 without using your Helm chart?

Yes, but we strongly recommend that you use the Helm chart to install if you want to leverage Lucidworks support.

Are there backup jobs used in disaster recovery situations for stateful services deployed to Kubernetes?

There is no out-of-the-box backup solution for stateful services provided with Fusion. Lucidworks recommends having a plan in place to reindex data from its primary source as part of an overall disaster recovery strategy.

Other questions?

If you have a question about a resource not included here, please reach out to Lucidworks support *(https://support.lucidworks.com/)*

Reference

The articles in this section describe configuration options for key Fusion components, API parameter names and values, and supported data types. The content is especially relevant to developers and search engineers who are configuring their search application.

Configuration Reference

Fusion's components are highly configurable. Each job, index and query pipeline stage, and parser has its own set of configuration options.

The articles in this section provide developer-focused details on what configuration options exist for each component.

Jobs

These reference topics give complete information about the configuration properties of jobs for which the subtype is "task" or "spark".

Jobs with the subtype "datasource" have configuration schemas that depend on the connector type. See *Connectors Configuration Reference*. For conceptual information and instructions for configuring and scheduling jobs, see *Jobs and Schedules*.

ALS Recommender Jobs

Use this job when you want to compute user recommendations or item similarities using a collaborative filtering recommender. You can also implement a user-to-item recommender in the advanced section of this job's configuration UI. This job uses SparkML's Alternating Least Squares (ALS) *(https://spark.apache.org/docs/2.1.0/ml-collaborative-filtering.html)*.

This job is deprecated. Use the *BPR Recommender* instead.

Default job name	`COLLECTION_NAME_item_recs`
Input	*Aggregated signals* (the `COLLECTION_NAME_signals_aggr` collections by default)
Output	• *Items-for-item recommendations* (the `COLLECTION_NAME_items_for_item_recommendations` collection by default) • *Items-for-user recommendations* (the `COLLECTION_NAME_items_for_user_recommendations` collection by default)

	query	count_i	type	timstamp_tdt	user_id	doc_id	session_id	fusion_query_id
Required signals fields:								

The `COLLECTION_NAME_user_item_prefs_agg` job provides input data for this job and must run before it. See *Built-in SQL Aggregation Jobs* for details.

This job assumes that your signals collection contains the preferences of many users. It uses this collection of preferences to predict another user's preference about an item that the user has not yet seen. A preference which can be viewed as a triple: user, item, and interaction-value.

When you *enable recommendations* for a collection, Fusion automatically creates an ALS Recommender job called `COLLECTION_NAME_item_recommendations`. This job generates both *items-for-user recommendations* and *items-for-item recommendations*, then stores the results in the `COLLECTION_NAME_items_for_user_recommendations` and `COLLECTION_NAME_items_for_item_recommendations` collections.

Basic job configuration

For items-for-item and items-for-user recommendations, the basic fields for configuring the `COLLECTION_NAME_item_recommendations` job are described below. To refine this job further, see *Advanced job configuration*.

- `numRecs`/**Number of User Recommendations to Compute**
  This is the number of recommendations that you want to return *per item* (for items-for-item recommendations) or *per user* (for items-for-user recommendations) in your dataset.

  Increasing this number up to 1000 will not cost too much computationally because the intensive work of computing the matrix decomposition (involving optimization) is already done by the time these recommendations are generated.

  Think of this as generating a matrix where the rows are the users and the columns are the recommendations. If we choose 1000 items to recommend, the size of the matrix will be `(number of users) x (number of items to recommend)`. For instance, if there are 10,000 users and 1000 recommendations, then the size of the matrix will be `10,000x1000`.

Input/output parameters

- `trainingCollection`/**Recommender Training Collection**
  Usually this should point to the `COLLECTION_NAME_recs_aggr` collection. If you are using another aggregated signals collection, verify that this field points to the correct collection name.
- `outputItemSimCollection`/**Item-to-item Similarity Collection**
  Usually this should point to the This collection will store the `N` most similar items for every item in the collection, where `N` is determined by the `numSims`/**Number of Item Similarities to Compute** field described below. Fusion can query this collection after the job to determine the most similar items to recommend based on an item choice.

	You can only specify a secondary collection of the collection with which this job is associated. For example, if you have a `Movies` collection and a `Films` collection and this job is associated with the `Movies` collection, then you cannot specify the `Films_items_for_item_recommendations` collection here.

Model tuning parameters

- `numSims`/**Number of Item Similarities to Compute**
  This is similar to `numRecs`/**Number of User Recommendations to Compute** in the sense that this number of similar items are found for each item in the collection. Think of it as a matrix of size: `(number of items) x (number of item similarities to compute)`.
  This is not computationally expensive because it is just a similarity calculation (which involves no optimization). A reasonable value would be 30–250. It will also depend on the number of items displayed in your search application.
- `implicitRatings`/**Implicit Preferences**
  The concept of Implicit preferences is explained in *Implicit vs explicit signals*.
  In this tutorial it is assumed that we submit no information about the items and the users (think of user and item features) but simply rely on the user-item interaction as a means to recommend similar products. That is the power of using implicit signals: we don't need to know information about the user or the item, just how much they interact with each other.
  If explicit ratings values are used (such as ratings from the user) then this box can be unchecked.
- `deleteOldRecs`/**Delete Old Recommendations**
  If you have reasons not to draw on old recommendations, then check this box. If this box is unchecked, then old recommendations will not be deleted but new recommendations will be appended with a different job ID. Both sets of recommendations will be contained within the same collection.

Advanced job configuration

You can achieve higher accuracy, and often reduce the training time too, by tuning the `COLLECTION_NAME_item_recommendations` job using the advanced configuration keys described here. In the job configuration panel, click **Advanced** to display these additional fields.

- `excludeFromDeleteFilter`/**Exclude from Delete Filter**
  If you have selected `deleteOldRecs`/**Delete Old Recommendations** but you do not want to completely delete all old recommendations, this field allows you to input a query that captures the data you want keep and removes the rest.
- `numUserRecsPerItem`/**Number of Users to Recommend to each Item**
  This setting indicates which users (from the known user group) are most likely to be interested in a particular item. The setting allows you to choose how many of the most interested users you would like to precompute and store.
  If one thinks of an estimated user-item matrix (after optimization), an item is a single column from the matrix, so if we wanted the top 100 users per item, we would sort the interest values in that column in descending order and take the top 100 row indexes which would correspond to individual users.
- `maxTrainingIterations`/**Maximum Training Iterations**
  The Alternating Least Squares algorithm involves optimization to find the two matrices `(user x latent factor and latent factor x item)` that best approximate the original user-item matrix (formed from the signals aggregation).
  The optimization occurs at the matrix entry level (every non-zero element) and it is iterative. Therefore, the more iterations that are allowed during optimization, the lower the cost function value, meaning more accurate hyperparameters which lead to better recommendations.
  However, the bigger the data, the longer the job takes to run because the number of constraints to satisfy have increased. A value of 10 iterations usually leads to effective results. Above a value of 15, the job will begin to slow dramatically for above 25 million signals.

Training data settings

- `trainingDataFilterQuery`/**Training Data Filter Query**
  This query setting is useful when the main signals collection does not have the *recommended fields*. The two most important fields are `doc_id` and `user_id` because the job must have a user-item pairing. Note that depending on how the signals are collected the names `doc_id` and `user_id` can be different, but the concept remains the same.
  There are times when not all the signals have these fields. In this case we can add a query to select a subset of data that does have a user-item pairing. It is done with the following query:

```
+doc_id:[* TO *] +user_id:[* TO *]
```

This query returns all signals documents that have a `user_id` and `doc_id` field. Each query is separated by a space. The plus (`+`) sign is a positive request for the field of interest, meaning return signals with `doc_id` instead of signals without `doc_id` (negated or opposite queries are returned by prefixing with a negative (`-`) sign).

- `popularItemMin`/**Training Data Filter By Popular Items**
  The underlying assumption of this parameter is that the more users that view an item, the more popular that item is. Therefore, this value signifies the minimum number of interactions that must occur with the item for it to be considered a training data point.

  The higher the number, the smaller amount of data available for training because it is unlikely that many users interacted with all of the items. However, the quality of the data will be higher.

  One way to speed up training is to increase this number along with the training data sampling fraction. A reasonable number is between 10 and 20 depending on the application and user base. For instance, a song may be played much more than a movie and both may have more interaction than purchasing an item.
- `trainingSampleFraction`/**Training Data Sampling Fraction**
  This value is the percentage of the signal data or training data that you want to use for training the recommender job. It is advised to set this value to 1 and reduce the training data size (while increasing quality) by increasing the **Training Data Filter By Popular Items** as well as increasing the weight threshold in the **Training Data Filter Query**.
- `userIdField`/**Training Collection User Id Field**
  The ALS algorithm needs users, items, and a score of their interaction. The user ID field is the field name within the signal data that represents a user ID.
- `itemIdField`/**Training Collection Item Id Field**
  The item ID field is the field name within the aggregated signal data that represents the item or documents of interest.
- `weightField`/**Training Collection Weight Field**
  The weight field contains the score representing the interest of the user in an item.
- `initialBlocks`/**Training Block Size**
  In *Spark*, the training data is split amongst the executors in unchangeable blocks. This parameter sets the size of these blocks for training, but it requires advanced knowledge of Spark internals. We recommend leaving this setting at -1.

Model settings

- `modelId`/**Recommender Model ID**
  The **Recommender Model ID** is assigned the field `modelId` in the `COLLECTION_NAME_items_for_item_recommendations` and `COLLECTION_NAME_items_for_user_recommendations` recommendations collections. This allows you to filter the recommendations by the recommender model ID. When the recommender job runs, a job ID is also assigned; therefore, you can see the results from different runs of the same job parameters. If you want to experiment with different parameters, it is advised to change the recommender model ID to reflect the parameters so that you can find the best parameters.
- `saveModel`/**Save Model in Solr**
  Saving the model in Solr adds the parameters to the `COLLECTION_NAME_recommender_models` collection as a document. Using this method allows you to track all the recommender configurations.
- `modelCollection`/**Model Collection**
  This is the collection to store the experiment configurations (`_recommender_models` by default).
- `alwaysTrain`/**Force model re-training**
  When the job runs, it checks to see whether the model ID for the job already exists in the model collection. If the model does exist, it uses the pre-existing model to get the recommendations. Otherwise, if the box is checked it will re-run the recommender job and redo the optimization from scratch. Unless you need to maintain this ID name, it is advisable to create a separate model ID for each new combination of parameters.

Grid search settings

- `initialRank`/**Recommender Rank**
  The recommender rank is the number of latent factors into which to decompose the original user-item matrix. A reasonable range is 50-200. Above 200, the performance of the optimization can degrade dramatically depending on computing resources.
- `gridSearchWidth`/**Grid Search Width**
  Grid search is an automatic way to determine the best parameters for the recommender model. It tries different combinations of parameters of equally spaced units within a parameter domain and takes the model that has the lowest cost function value. This is a long process because a single run can take several hours depending on the computing resources, so trying multiple combinations can take some time. Depending on the size of your training data, it is better to do a manual grid search to reduce the number of runs needed to find a suitable recommender model.
- `initialAlpha`/**Implicit Preference Confidence**
  The implicit preference confidence is an approximation of how confident you are that the implicit data does indeed represent an accurate level of interest of a user in an item. Typical values are 1-100, with 100 being more confident in the training data

representing the interest of the user. This parameter is used as a regularizer for optimization. The higher the confidence value, the more the optimization is penalized for a wrong approximation of the interest value.

- `initialLambda`/**Initial Lambda**
  Lambda is another optimization parameter that prevents overfitting. Remember we are decomposing the user-item matrix by estimating two matrices. The values in these matrices can be any number, large or small, and have a wide spread in the values themselves. To keep the scale of the value consistent or reduce the spread of the values, we use a regularizer. The higher the lambda, the smaller the values in the two estimated matrices. A smaller lambda gives the algorithm more freedom to estimate an answer which can result in overfitting. Typical values are between 0.01 and 0.3.
- `randomSeed`/**Random Seed**
  When the two matrices are first being estimated, their values are set randomly as an initialization. As the optimization proceeds the values are changed according to the error in the optimization. When training it is important to keep the initialization the same in order to determine the effect of different values of parameters in the model. Keep this value the same across all experiments.

Item metadata settings

- `itemMetadataCollection`/**Item Metadata Collection**
  The main collection has very detailed information about each item, much of which is not necessary for training the recommender system. All that is important to train the recommender are the document IDs and the known users. If you have this metadata in a different collection than the main collection, enter that collection's name here. Once the training is complete, the document ID of the relevant documents can be used to retrieve detailed information from the item catalog. The point is to train on small data per item and retrieve the detailed information for only relevant documents.
- `itemMetadataJoinField`/**Item Metadata Join Field**
  This is the field that is common to the aggregated signal data and the original data. It is used to join each document from the recommender collection to the original item in the main collection. Usually this is the `id` field.
- `itemMetadataFields`/**Item Metadata Fields**
  These are fields from the main collection that should be returned with each recommendation. You can add fields here by clicking the **Add** icon. To ensure that this works correctly, verify that `itemMetadataJoinField`/**Item Metadata Join Field** has the correct value.

Items-for-Item Recommendations Configuration (ALS)

Basic job configuration

For items-for-item and items-for-user recommendations, the basic fields for configuring the `COLLECTION_NAME_item_recommendations` job are described below. To refine this job further, see *Advanced job configuration*.

- `numRecs`/**Number of User Recommendations to Compute**
  This is the number of recommendations that you want to return *per item* (for items-for-item recommendations) or *per user* (for items-for-user recommendations) in your dataset.

  Increasing this number up to 1000 will not cost too much computationally because the intensive work of computing the matrix decomposition (involving optimization) is already done by the time these recommendations are generated.

  Think of this as generating a matrix where the rows are the users and the columns are the recommendations. If we choose 1000 items to recommend, the size of the matrix will be `(number of users) x (number of items to recommend)`. For instance, if there are 10,000 users and 1000 recommendations, then the size of the matrix will be `10,000x1000`.

Input/output parameters

- `trainingCollection`/**Recommender Training Collection**
  Usually this should point to the `COLLECTION_NAME_recs_aggr` collection. If you are using another aggregated signals collection, verify that this field points to the correct collection name.
- `outputItemSimCollection`/**Item-to-item Similarity Collection**
  Usually this should point to the `COLLECTION_NAME_items_for_item_recommendations` collection. `COLLECTION_NAME_items_for_user_recommendations` collection. This collection will store the `N` most similar items for every item in the collection, where `N` is determined by the `numSims`/**Number of Item Similarities to Compute** field described below. Fusion can query this collection after the job to determine the most similar items to recommend based on an item choice.

  > You can only specify a secondary collection of the collection with which this job is associated. For example, if you have a `Movies` collection and a `Films` collection and this job is associated with the `Movies` collection, then you cannot specify the `Films_items_for_item_recommendations` collection here.

Model tuning parameters

- `numSims`/**Number of Item Similarities to Compute**
  This is similar to `numRecs`/**Number of User Recommendations to Compute** in the sense that this number of similar items are found for each item in the collection. Think of it as a matrix of size: `(number of items) x (number of item similarities to compute)`.

  This is not computationally expensive because it is just a similarity calculation (which involves no optimization). A reasonable value would be 30–250. It will also depend on the number of items displayed in your search application.
- `implicitRatings`/**Implicit Preferences**
  The concept of Implicit preferences is explained in *Implicit vs explicit signals*.

  In this tutorial it is assumed that we submit no information about the items and the users (think of user and item features) but simply rely on the user-item interaction as a means to recommend similar products. That is the power of using implicit signals: we don't need to know information about the user or the item, just how much they interact with each other.

  If explicit ratings values are used (such as ratings from the user) then this box can be unchecked.
- `deleteOldRecs`/**Delete Old Recommendations**
  If you have reasons not to draw on old recommendations, then check this box. If this box is unchecked, then old recommendations will not be deleted but new recommendations will be appended with a different job ID. Both sets of recommendations will be contained within the same collection.

Advanced job configuration

You can achieve higher accuracy, and often reduce the training time too, by tuning the `COLLECTION_NAME_item_recommendations` job using the advanced configuration keys described here. In the job configuration panel, click **Advanced** to display these additional fields.

- `excludeFromDeleteFilter`/**Exclude from Delete Filter**
  If you have selected `deleteOldRecs`/**Delete Old Recommendations** but you do not want to completely delete all old recommendations, this field allows you to input a query that captures the data you want keep and removes the rest.
- `numUserRecsPerItem`/**Number of Users to Recommend to each Item**
  This setting indicates which users (from the known user group) are most likely to be interested in a particular item. The setting allows you to choose how many of the most interested users you would like to precompute and store.

If one thinks of an estimated user-item matrix (after optimization), an item is a single column from the matrix, so if we wanted the top 100 users per item, we would sort the interest values in that column in descending order and take the top 100 row indexes which would correspond to individual users.

- `maxTrainingIterations`/**Maximum Training Iterations**
  The Alternating Least Squares algorithm involves optimization to find the two matrices `(user x latent factor and latent factor x item)` that best approximate the original user-item matrix (formed from the signals aggregation).

  The optimization occurs at the matrix entry level (every non-zero element) and it is iterative. Therefore, the more iterations that are allowed during optimization, the lower the cost function value, meaning more accurate hyperparameters which lead to better recommendations.

  However, the bigger the data, the longer the job takes to run because the number of constraints to satisfy have increased. A value of 10 iterations usually leads to effective results. Above a value of 15, the job will begin to slow dramatically for above 25 million signals.

Training data settings

- `trainingDataFilterQuery`/**Training Data Filter Query**
  This query setting is useful when the main signals collection does not have the *recommended fields*. The two most important fields are `doc_id` and `user_id` because the job must have a user-item pairing. Note that depending on how the signals are collected the names `doc_id` and `user_id` can be different, but the concept remains the same.

  There are times when not all the signals have these fields. In this case we can add a query to select a subset of data that does have a user-item pairing. It is done with the following query:

  ```
  +doc_id:[* TO *] +user_id:[* TO *]
  ```

  This query returns all signals documents that have a `user_id` and `doc_id` field. Each query is separated by a space. The plus (`+`) sign is a positive request for the field of interest, meaning return signals with `doc_id` instead of signals without `doc_id` (negated or opposite queries are returned by prefixing with a negative (`-`) sign).
- `popularItemMin`/**Training Data Filter By Popular Items**
  The underlying assumption of this parameter is that the more users that view an item, the more popular that item is. Therefore, this value signifies the minimum number of interactions that must occur with the item for it to be considered a training data point.

  The higher the number, the smaller amount of data available for training because it is unlikely that many users interacted with all of the items. However, the quality of the data will be higher.

  One way to speed up training is to increase this number along with the training data sampling fraction. A reasonable number is between 10 and 20 depending on the application and user base. For instance, a song may be played much more than a movie and both may have more interaction than purchasing an item.
- `trainingSampleFraction`/**Training Data Sampling Fraction**
  This value is the percentage of the signal data or training data that you want to use for training the recommender job. It is advised to set this value to 1 and reduce the training data size (while increasing quality) by increasing the **Training Data Filter By Popular Items** as well as increasing the weight threshold in the **Training Data Filter Query**.
- `userIdField`/**Training Collection User Id Field**
  The ALS algorithm needs users, items, and a score of their interaction. The user ID field is the field name within the signal data that represents a user ID.
- `itemIdField`/**Training Collection Item Id Field**
  The item ID field is the field name within the aggregated signal data that represents the item or documents of interest.
- `weightField`/**Training Collection Weight Field**
  The weight field contains the score representing the interest of the user in an item.
- `initialBlocks`/**Training Block Size**
  In *Spark*, the training data is split amongst the executors in unchangeable blocks. This parameter sets the size of these blocks for training, but it requires advanced knowledge of Spark internals. We recommend leaving this setting at -1.

Model settings

- `modelId`/**Recommender Model ID**
  The **Recommender Model ID** is assigned the field `modelId` in the `COLLECTION_NAME_items_for_item_recommendations` and `COLLECTION_NAME_items_for_user_recommendations` recommendations collections. This allows you to filter the recommendations by the recommender model ID. When the recommender job runs, a job ID is also assigned; therefore, you can see

the results from different runs of the same job parameters. If you want to experiment with different parameters, it is advised to change the recommender model ID to reflect the parameters so that you can find the best parameters.

- `saveModel`/**Save Model in Solr**
  Saving the model in Solr adds the parameters to the `COLLECTION_NAME_recommender_models` collection as a document. Using this method allows you to track all the recommender configurations.
- `modelCollection`/**Model Collection**
  This is the collection to store the experiment configurations (`_recommender_models` by default).
- `alwaysTrain`/**Force model re-training**
  When the job runs, it checks to see whether the model ID for the job already exists in the model collection. If the model does exist, it uses the pre-existing model to get the recommendations. Otherwise, if the box is checked it will re-run the recommender job and redo the optimization from scratch. Unless you need to maintain this ID name, it is advisable to create a separate model ID for each new combination of parameters.

Grid search settings

- `initialRank`/**Recommender Rank**
  The recommender rank is the number of latent factors into which to decompose the original user-item matrix. A reasonable range is 50-200. Above 200, the performance of the optimization can degrade dramatically depending on computing resources.
- `gridSearchWidth`/**Grid Search Width**
  Grid search is an automatic way to determine the best parameters for the recommender model. It tries different combinations of parameters of equally spaced units within a parameter domain and takes the model that has the lowest cost function value. This is a long process because a single run can take several hours depending on the computing resources, so trying multiple combinations can take some time. Depending on the size of your training data, it is better to do a manual grid search to reduce the number of runs needed to find a suitable recommender model.
- `initialAlpha`/**Implicit Preference Confidence**
  The implicit preference confidence is an approximation of how confident you are that the implicit data does indeed represent an accurate level of interest of a user in an item. Typical values are 1-100, with 100 being more confident in the training data representing the interest of the user. This parameter is used as a regularizer for optimization. The higher the confidence value, the more the optimization is penalized for a wrong approximation of the interest value.
- `initialLambda`/**Initial Lambda**
  Lambda is another optimization parameter that prevents overfitting. Remember we are decomposing the user-item matrix by estimating two matrices. The values in these matrices can be any number, large or small, and have a wide spread in the values themselves. To keep the scale of the value consistent or reduce the spread of the values, we use a regularizer. The higher the lambda, the smaller the values in the two estimated matrices. A smaller lambda gives the algorithm more freedom to estimate an answer which can result in overfitting. Typical values are between 0.01 and 0.3.
- `randomSeed`/**Random Seed**
  When the two matrices are first being estimated, their values are set randomly as an initialization. As the optimization proceeds the values are changed according to the error in the optimization. When training it is important to keep the initialization the same in order to determine the effect of different values of parameters in the model. Keep this value the same across all experiments.

Item metadata settings

- `itemMetadataCollection`/**Item Metadata Collection**
  The main collection has very detailed information about each item, much of which is not necessary for training the recommender system. All that is important to train the recommender are the document IDs and the known users. If you have this metadata in a different collection than the main collection, enter that collection's name here. Once the training is complete, the document ID of the relevant documents can be used to retrieve detailed information from the item catalog. The point is to train on small data per item and retrieve the detailed information for only relevant documents.
- `itemMetadataJoinField`/**Item Metadata Join Field**
  This is the field that is common to the aggregated signal data and the original data. It is used to join each document from the recommender collection to the original item in the main collection. Usually this is the `id` field.
- `itemMetadataFields`/**Item Metadata Fields**
  These are fields from the main collection that should be returned with each recommendation. You can add fields here by clicking the **Add** icon. To ensure that this works correctly, verify that `itemMetadataJoinField`/**Item Metadata Join Field** has the correct value.

Items-for-User Recommendations Configuration (ALS)

Basic job configuration

For items-for-item and items-for-user recommendations, the basic fields for configuring the `COLLECTION_NAME_item_recommendations` job are described below. To refine this job further, see *Advanced job configuration*.

- `numRecs`/**Number of User Recommendations to Compute**
  This is the number of recommendations that you want to return *per item* (for items-for-item recommendations) or *per user* (for items-for-user recommendations) in your dataset.

  Increasing this number up to 1000 will not cost too much computationally because the intensive work of computing the matrix decomposition (involving optimization) is already done by the time these recommendations are generated.

  Think of this as generating a matrix where the rows are the users and the columns are the recommendations. If we choose 1000 items to recommend, the size of the matrix will be `(number of users) x (number of items to recommend)`. For instance, if there are 10,000 users and 1000 recommendations, then the size of the matrix will be `10,000x1000`.

Input/output parameters

- `trainingCollection`/**Recommender Training Collection**
  Usually this should point to the `COLLECTION_NAME_recs_aggr` collection. If you are using another aggregated signals collection, verify that this field points to the correct collection name.
- `outputItemSimCollection`/**Item-to-item Similarity Collection**
  Usually this should point to the `COLLECTION_NAME_items_for_item_recommendations` collection. `COLLECTION_NAME_items_for_user_recommendations` collection. This collection will store the `N` most similar items for every item in the collection, where `N` is determined by the `numSims`/**Number of Item Similarities to Compute** field described below. Fusion can query this collection after the job to determine the most similar items to recommend based on an item choice.

  > You can only specify a secondary collection of the collection with which this job is associated. For example, if you have a `Movies` collection and a `Films` collection and this job is associated with the `Movies` collection, then you cannot specify the `Films_items_for_item_recommendations` collection here.

Model tuning parameters

- `numSims`/**Number of Item Similarities to Compute**
  This is similar to `numRecs`/**Number of User Recommendations to Compute** in the sense that this number of similar items are found for each item in the collection. Think of it as a matrix of size: `(number of items) x (number of item similarities to compute)`.

  This is not computationally expensive because it is just a similarity calculation (which involves no optimization). A reasonable value would be 30–250. It will also depend on the number of items displayed in your search application.
- `implicitRatings`/**Implicit Preferences**
  The concept of Implicit preferences is explained in *Implicit vs explicit signals*.

  In this tutorial it is assumed that we submit no information about the items and the users (think of user and item features) but simply rely on the user-item interaction as a means to recommend similar products. That is the power of using implicit signals: we don't need to know information about the user or the item, just how much they interact with each other.

  If explicit ratings values are used (such as ratings from the user) then this box can be unchecked.
- `deleteOldRecs`/**Delete Old Recommendations**
  If you have reasons not to draw on old recommendations, then check this box. If this box is unchecked, then old recommendations will not be deleted but new recommendations will be appended with a different job ID. Both sets of recommendations will be contained within the same collection.

Advanced job configuration

You can achieve higher accuracy, and often reduce the training time too, by tuning the `COLLECTION_NAME_item_recommendations` job using the advanced configuration keys described here. In the job configuration panel, click **Advanced** to display these additional fields.

- `excludeFromDeleteFilter`/**Exclude from Delete Filter**
  If you have selected `deleteOldRecs`/**Delete Old Recommendations** but you do not want to completely delete all old recommendations, this field allows you to input a query that captures the data you want keep and removes the rest.
- `numUserRecsPerItem`/**Number of Users to Recommend to each Item**
  This setting indicates which users (from the known user group) are most likely to be interested in a particular item. The setting allows you to choose how many of the most interested users you would like to precompute and store.

If one thinks of an estimated user-item matrix (after optimization), an item is a single column from the matrix, so if we wanted the top 100 users per item, we would sort the interest values in that column in descending order and take the top 100 row indexes which would correspond to individual users.

- `maxTrainingIterations`/**Maximum Training Iterations**
  The Alternating Least Squares algorithm involves optimization to find the two matrices `(user x latent factor and latent factor x item)` that best approximate the original user-item matrix (formed from the signals aggregation).

  The optimization occurs at the matrix entry level (every non-zero element) and it is iterative. Therefore, the more iterations that are allowed during optimization, the lower the cost function value, meaning more accurate hyperparameters which lead to better recommendations.

  However, the bigger the data, the longer the job takes to run because the number of constraints to satisfy have increased. A value of 10 iterations usually leads to effective results. Above a value of 15, the job will begin to slow dramatically for above 25 million signals.

Training data settings

- `trainingDataFilterQuery`/**Training Data Filter Query**
  This query setting is useful when the main signals collection does not have the *recommended fields*. The two most important fields are `doc_id` and `user_id` because the job must have a user-item pairing. Note that depending on how the signals are collected the names `doc_id` and `user_id` can be different, but the concept remains the same.

  There are times when not all the signals have these fields. In this case we can add a query to select a subset of data that does have a user-item pairing. It is done with the following query:

  ```
  +doc_id:[* TO *] +user_id:[* TO *]
  ```

  This query returns all signals documents that have a `user_id` and `doc_id` field. Each query is separated by a space. The plus (`+`) sign is a positive request for the field of interest, meaning return signals with `doc_id` instead of signals without `doc_id` (negated or opposite queries are returned by prefixing with a negative (`-`) sign).
- `popularItemMin`/**Training Data Filter By Popular Items**
  The underlying assumption of this parameter is that the more users that view an item, the more popular that item is. Therefore, this value signifies the minimum number of interactions that must occur with the item for it to be considered a training data point.

  The higher the number, the smaller amount of data available for training because it is unlikely that many users interacted with all of the items. However, the quality of the data will be higher.

  One way to speed up training is to increase this number along with the training data sampling fraction. A reasonable number is between 10 and 20 depending on the application and user base. For instance, a song may be played much more than a movie and both may have more interaction than purchasing an item.
- `trainingSampleFraction`/**Training Data Sampling Fraction**
  This value is the percentage of the signal data or training data that you want to use for training the recommender job. It is advised to set this value to 1 and reduce the training data size (while increasing quality) by increasing the **Training Data Filter By Popular Items** as well as increasing the weight threshold in the **Training Data Filter Query**.
- `userIdField`/**Training Collection User Id Field**
  The ALS algorithm needs users, items, and a score of their interaction. The user ID field is the field name within the signal data that represents a user ID.
- `itemIdField`/**Training Collection Item Id Field**
  The item ID field is the field name within the aggregated signal data that represents the item or documents of interest.
- `weightField`/**Training Collection Weight Field**
  The weight field contains the score representing the interest of the user in an item.
- `initialBlocks`/**Training Block Size**
  In *Spark*, the training data is split amongst the executors in unchangeable blocks. This parameter sets the size of these blocks for training, but it requires advanced knowledge of Spark internals. We recommend leaving this setting at -1.

Model settings

- `modelId`/**Recommender Model ID**
  The **Recommender Model ID** is assigned the field `modelId` in the `COLLECTION_NAME_items_for_item_recommendations` and `COLLECTION_NAME_items_for_user_recommendations` recommendations collections. This allows you to filter the recommendations by the recommender model ID. When the recommender job runs, a job ID is also assigned; therefore, you can see

the results from different runs of the same job parameters. If you want to experiment with different parameters, it is advised to change the recommender model ID to reflect the parameters so that you can find the best parameters.

- `saveModel`/**Save Model in Solr**
  Saving the model in Solr adds the parameters to the `COLLECTION_NAME_recommender_models` collection as a document. Using this method allows you to track all the recommender configurations.
- `modelCollection`/**Model Collection**
  This is the collection to store the experiment configurations (`_recommender_models` by default).
- `alwaysTrain`/**Force model re-training**
  When the job runs, it checks to see whether the model ID for the job already exists in the model collection. If the model does exist, it uses the pre-existing model to get the recommendations. Otherwise, if the box is checked it will re-run the recommender job and redo the optimization from scratch. Unless you need to maintain this ID name, it is advisable to create a separate model ID for each new combination of parameters.

Grid search settings

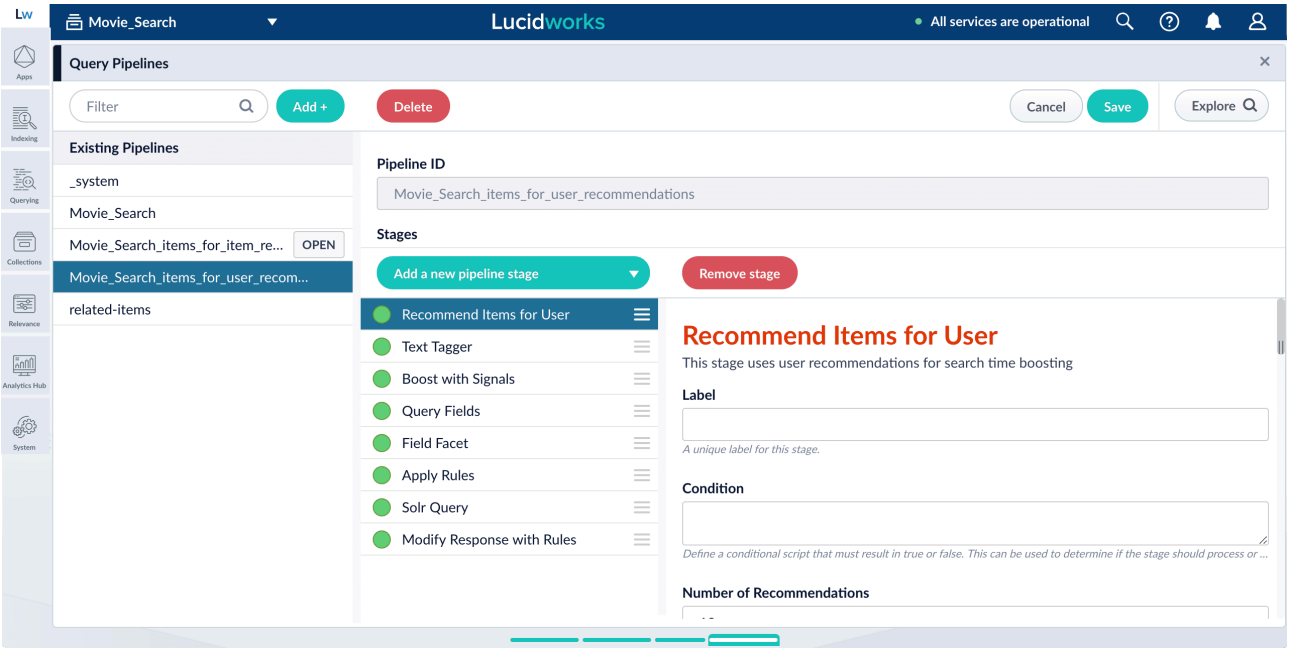
- `initialRank`/**Recommender Rank**
  The recommender rank is the number of latent factors into which to decompose the original user-item matrix. A reasonable range is 50-200. Above 200, the performance of the optimization can degrade dramatically depending on computing resources.
- `gridSearchWidth`/**Grid Search Width**
  Grid search is an automatic way to determine the best parameters for the recommender model. It tries different combinations of parameters of equally spaced units within a parameter domain and takes the model that has the lowest cost function value. This is a long process because a single run can take several hours depending on the computing resources, so trying multiple combinations can take some time. Depending on the size of your training data, it is better to do a manual grid search to reduce the number of runs needed to find a suitable recommender model.
- `initialAlpha`/**Implicit Preference Confidence**
  The implicit preference confidence is an approximation of how confident you are that the implicit data does indeed represent an accurate level of interest of a user in an item. Typical values are 1-100, with 100 being more confident in the training data representing the interest of the user. This parameter is used as a regularizer for optimization. The higher the confidence value, the more the optimization is penalized for a wrong approximation of the interest value.
- `initialLambda`/**Initial Lambda**
  Lambda is another optimization parameter that prevents overfitting. Remember we are decomposing the user-item matrix by estimating two matrices. The values in these matrices can be any number, large or small, and have a wide spread in the values themselves. To keep the scale of the value consistent or reduce the spread of the values, we use a regularizer. The higher the lambda, the smaller the values in the two estimated matrices. A smaller lambda gives the algorithm more freedom to estimate an answer which can result in overfitting. Typical values are between 0.01 and 0.3.
- `randomSeed`/**Random Seed**
  When the two matrices are first being estimated, their values are set randomly as an initialization. As the optimization proceeds the values are changed according to the error in the optimization. When training it is important to keep the initialization the same in order to determine the effect of different values of parameters in the model. Keep this value the same across all experiments.

Item metadata settings

- `itemMetadataCollection`/**Item Metadata Collection**
  The main collection has very detailed information about each item, much of which is not necessary for training the recommender system. All that is important to train the recommender are the document IDs and the known users. If you have this metadata in a different collection than the main collection, enter that collection's name here. Once the training is complete, the document ID of the relevant documents can be used to retrieve detailed information from the item catalog. The point is to train on small data per item and retrieve the detailed information for only relevant documents.
- `itemMetadataJoinField`/**Item Metadata Join Field**
  This is the field that is common to the aggregated signal data and the original data. It is used to join each document from the recommender collection to the original item in the main collection. Usually this is the `id` field.
- `itemMetadataFields`/**Item Metadata Fields**
  These are fields from the main collection that should be returned with each recommendation. You can add fields here by clicking the **Add** icon. To ensure that this works correctly, verify that `itemMetadataJoinField`/**Item Metadata Join Field** has the correct value.

Query pipeline configuration

If you have enabled signals and recommendations for a collection, then the default `COLLECTION_NAME_items_for_user_recommendations` query pipeline is already created and configured to serve items-for-user recommendations:



You can tune the *Recommend Items for User* query stage as needed to refine the output.

Aggregation Jobs

Aggregation jobs compile your raw signals into aggregated signals. Most Fusion jobs that consume signals require aggregated signals.

Default job name	`COLLECTION_NAME_click_signals_aggregation`
Input	Raw signals (the `COLLECTION_NAME_signals` collection by default)
Output	*Aggregated signals* (the `COLLECTION_NAME_signals_aggr` collection by default)

	query	count_i	type	timstamp_tdt	user_id	doc_id	session_id	fusion_query_id
Required signals fields:								[*1*]

1 Required if you are using response signals.

Aggregation properties

The aggregation process is specified by an aggregation type consisting of the following list of properties:

Name	Description
`id`	Aggregation ID
`groupingFields`	List of signal field names
`signalTypes`	List of signal types
`aggregator`	Symbolic name of the aggregator implementation
`selectQuery`	Query string, default `*:*`
`sort`	Ordering of aggregated signals
`timeRange`	String specifying time range, e.g., `[* TO NOW]`
`outputPipeline`	Pipeline ID for processing aggregated events
`outputCollection`	Output collection name
`rollupPipeline`	Rollup pipeline ID
`rollupAggregator`	Name of the aggregator implementation used for rollups
`sourceRemove`	Boolean, default is false
`sourceCatchup`	Boolean, default is true
`outputRollup`	Boolean, default is true
`aggregates`	List of aggregation functions
`params`	Arbitrary parameters to be used by specific aggregator implementations

Configuration properties

Aggregator Functions Examples

Arithmetic Functions

Arithmetic functions operate on all valid numeric values (including string fields that are parseable into double numbers) from source fields and compute a single result to the target field.

sum

A sum of numeric values, as a double number.

Example:

```
{ "type" : "sum", "sourceFields" : [ "count_i" ], "targetField" : "sum_count_d" }
```

sumOfLogs

A sum of natural logs of numeric value, as a double number.

Example:

```
{ "type": "sumOfLogs", "sourceFields": [ "script_d" ], "targetField":
"script_sum_logs_d" }
```

sumOfSquares

A sum of squares of numeric value, as a double number.

Example:

```
{ "type" : "sumOfSquares", "sourceFields" : [ "params.position_s" ], "targetField"
: "sumOfSquares_position_d" }
```

count

A count of source values, as a long number.

Example:

```
{ "type": "count", "sourceFields": [ "id" ], "targetField": "count_d" }
```

geoMean

A geometric mean of numeric values, as a double number.

Example:

```
{ "type" : "geoMean", "sourceFields" : [ "params.position_s" ], "targetField" :
"geoMean_position_d" }
```

mean

An arithmetic mean of numeric values, as a double number.

Example:

```
{ "type" : "mean", "sourceFields" : [ "params.position_s" ], "targetField" :
"mean_position_d" }
```

min

The minimum numeric value.

Example:

```
{ "type" : "min", "sourceFields" : [ "params.position_s" ], "targetField" :
"min_position_d" }
```

max

The maximum numeric value.

```
{ "type" : "max", "sourceFields" : [ "params.position_s" ], "targetField" :
"max_position_d" }
```

decay_sum

A sum of time-based exponentially decayed numeric values. The difference between the aggregationTime and the event time will be decayed using an exponential function with a half-life equaling 30 days.

This function has some additional properties:

- halfLife: the number of seconds for the half-life decay function.
- timestampField: the name of the field that contains the source event's timestamp. By default, this is `timestamp_dt`.
- defaultWeight: the weight of an event if all values from source fields are missing. The default is 0.1f, and this is expressed as a float.

Example:

```
{ "type" : "decay_sum", "sourceFields" : [ "weight_d" ], "targetField" :
"decay_sum_weight_d", "params" : { } }
```

String Functions

String functions operate all values from source fields treated as strings.

cat

A concatenation of string values.

This function has some additional properties:

- separator: the character to use as a delimiter between values. The default is a single space.
- maxStringLength: the maximum length of the concatenated values (including separators). When this limit is exceeded, additional values are discarded. The default value is 10485760 characters (10 * 1024 * 1024).
- maxValueCount: the maximum number of values to concatenate. Any values collected after this limit are discarded. The default is 100.

Example:

```
{ "type" : "cat", "sourceFields" : [ "user_id_s" ], "targetField" :
"cat_user_id_txt", "params" : { } }
```

ucat

A concatenation of unique string values.

This function has some additional properties:

- separator: the character to use as a delimiter between values. The default is a single space.
- maxStringLength: the maximum length of the concatenated values (including separators). When this limit is exceeded, additional values are discarded. The default value is 10485760 characters (10 * 1024 * 1024).
- maxValueCount: the maximum number of values to concatenate. Any values collected after this limit are discarded. The default is 100.

Example:

```
{ "type" : "ucat", "sourceFields" : [ "user_id_s" ], "targetField" :
"ucat_user_id_txt", "params" : { } }
```

split

A simple regex-based string splitting function.

The following function params are supported:

- regex. (string, required) a regular expression used for splitting.
- lower. (boolean, optional, false by default) after the regex has been applied the resulting parts are optionally lower-cased (using US locale).

Example:

```
{ "type" : "split", "sourceFields" : [ "query_s" ], "targetField" :
"event:query_split", "params" : { "regex": "\\s+", "lower": true } }
```

In the example above, the raw signal event field "query_s" is first split on whitespace and then lower-cased, and the result is put back into the raw signal event field "query_split".

replace

A simple regex-based string replace. The java.util.regex.Pattern syntax is supported for the regex matching and replacement.

The following function params are supported:

- regex. (string, required) a pattern to match.
- replace. (string, required) replacement.

Example:

```
{ "type" : "replace", "sourceFields" : [ "query_split" ], "targetField" :
"event:query_split_clean", "params" : { "regex": "\\P{Alpha}+", "replace": "_" } }
```

In the example above, this function takes the "query_split" values and replaces all non-alphabetic characters with underscores, and stores the result in the event's field "query_split_clean". As an extended example, this function would follow after the example **split** function and would add the field "query_split_clean" to the raw signal event. The "query_split_clean" field could be aggregated via other aggregation functions.

Collection Functions

Collection functions simply collect values from the source fields and add them as multiple values to the target field.

discard

This function discards all values from source fields and the target field. This modifies the source event and any in-progress aggregation result. This creates side-effects for subsequent functions, so should be used with care.

Example:

```
{ "type" : "discard", "sourceFields" : [ "user_id_s" ], "targetField" :
"collect_user_id_ss", "params" : { } }
```

collect

Collect values from source fields.

This function has one additional property, 'maxValueCount', which defines the number of fields to collect from source fields. Any fields collected after this limit are discarded. The default is 100.

Example:

```
{ "type" : "collect", "sourceFields" : [ "user_id_s" ], "targetField" :
"collect_user_id_ss", "params" : { } }
```

ucollectCollect unique values from source fields.

This function has one additional property, 'maxValueCount', which defines the number of fields to collect from source fields. Any fields collected after this limit are discarded. The default is 100.

Example:

```
{ "type" : "ucollect", "sourceFields" : [ "user_id_s" ], "targetField" :
"unique_user_id_ss", "params" : { } }
```

Statistical Functions

Statistical functions compute scalar and matrix statistics. When the function has multiple results, such as for matrix or vector results, the data is stored in multiple fields.

varianceThe square of standard deviation of numeric values, as a double number.

Example:

```
{ "type" : "covariance", "sourceFields" : [ "params.position_s", "position_rnd_1",
"position_rnd_2" ], "targetField" : "cov_position_d", "params" : { } }
```

stddev

The standard deviation of numeric values, as a double number.

Example:

```
{ "type" : "stddev", "sourceFields" : [ "params.position_s" ], "targetField" :
"stddev_position_d", "params" : { } }
```

cardinality

An estimate of the number of unique elements in the set of values from source fields (which are treated as strings). This uses the HyperLogLog implementation.

This function has one additional property, 'error', which defines the acceptable probability of error from real value, specifically the standard deviation from real results. Smaller values cause exponentially higher RAM consumption during processing. For example, the default, 0.1, uses ~8Kb of RAM, while tests have shown 0.0001 uses ~64Mb.

Example:

```
{ "type" : "cardinality", "sourceFields" : [ "params.position_s" ], "targetField" :
"cardinality_position_l", "params" : { } }
```

skewness

The measure of asymmetry of the distribution around its mean. This function is performed on numeric values and is expressed as a double number.

Example:

```
{ "type" : "skewness", "sourceFields" : [ "params.position_s" ], "targetField" :
"skewness_position_d", "params" : { } }
```

kurtosis

The adjusted Pearson's kurtosis of numeric values, expressed as a double. This provides a comparison of the shape of the distribution to that of the normal distribution.

Example:

```
{ "type" : "kurtosis", "sourceFields" : [ "params.position_s" ], "targetField" :
"kurtosis_position_d", "params" : { } }
```

quantiles

The quantiles of numeric values, stored as a double number in 0-N.targetField, or as a list of values in the target field (depending on the 'multivalued' property, described below). This implementation uses the T-Digest structure.

This function has the following additional properties:

- quantiles: the number of quantiles. The default is 10.
- multiValued: when true, all quantiles will be stored as multiple values in the target field. If false, then multiple values will be created in the format '0.targetField' to 'N.targetField'.

Example:

```
{ "type" : "quantiles", "sourceFields" : [ "params.position_s" ], "targetField" :
"quantiles_position_ss", "params" : { "multiValued" : true } }
```

topk

An estimate of the top-K elements in the source fields and their frequency. The result is stored in three multi-valued fields, each with the same number of values. The three fields are:

- counts.targetField: integer counts (frequencies) of elements.
- values.targetField: elements.
- errors.targetField: estimation errors.

This function has one additional property, 'k', which is the number of elements to report. The default is 10.

Example:

```
{ "type" : "topk", "sourceFields" : [ "params.position_s" ], "targetField" :
"topk_position_ss", "params" : { } }
```

covariance

A covariance matrix of numeric values from N > 1 source fields, with no smoothing. Missing or invalid values are treated as 0.0. A row of missing values is ignored. The resulting covariance matrix is stored in `N * (N - 1)` fields following the naming pattern 'sourceField1.sourceField2.targetField'.

If source fields contain multiple values, only the first value from each source field will be used.

This implementation runs in a constant and small memory budget.

Example:

```
{ "type" : "covariance", "sourceFields" : [ "params.position_s", "position_rnd_1",
"position_rnd_2" ], "targetField" : "cov_position_d", "params" : { } }
```

correlation

A correlation matrix of numeric values from N > 1 source fields. This implementation is based on the covariance function. The resulting correlation matrix is stored in `N * (N - 1)` fields following the naming pattern 'sourceField1.sourceField2.targetField'.

Example:

```
{ "type" : "correlation", "sourceFields" : [ "params.position_s", "position_rnd_1",
"position_rnd_2" ], "targetField" : "corr_position_d", "params" : { } }
```

histogram

An approximate histogram of values and their counts in source fields, using the T-Digest algorithm. Results are stored as corresponding multiple values in 'means.targetField' (for double values) and 'counts.targetField' (for integer values).

Example:

```
{ "type" : "histogram", "sourceFields" : [ "params.position_s" ], "targetField" :
"histogram_position_ss", "params" : { } }
```

sigmoid

This function uses hyperbolic tangent (tanh) to limit the impact of source values according to an s-shaped curve. The following parameters control the shape of the curve:

- weight. controls the range of values. Default weight is 1.0, which means that the sigmoid function values will range between (-1, 1). E.g. weight = 2.0 means that values will range between (-2, 2).
- intercept. sets the constant shift of function values. Default is 0, which means that sigmoid(0) = 0 and sigmoid(Inf) → 1. E.g. intercept = 2.0 means that sigmoid(0) = 2.0 and sigmoid(Inf) = 3.0.
- slope. this parameter controls the slope of the function, i.e. how quickly it reaches saturation. Default value is 1.0. E.g. slope = 2 will cause the function to saturate quickly, slope = 0.1 will cause the function to saturate for larger values of source.
- final. boolean, default is true. This controls how the sigmoid is applied to the source value. First, all numeric values from source fields are summed. Then, if final = false the current sum is passed to the sigmoid function and added to the previous total. If final = true then the current sum is added to the total and the sigmoid function is applied only at the end of the aggregation.

Example:

```
{ "type" : "sigmoid", "sourceFields" : [ "params.position_s" ], "targetField" :
"sigmoid_position_ss", "params" : { "weight" : 2.0, "intercept" : 10.0, "slope" :
0.5, "final" : true } }
```

Logical Functions

when

A logical function where processing will continue only if this function evaluates to true.

This function takes one additional property, 'expr', which is a JavaScript expression that must evaluate to a Boolean true/false. This property takes the same objects as the 'expr' function, described above. If this property is missing, the function will evaluate to true when any sourceField or targetField is present.

Example:

```
{ "type" : "when", "sourceFields" : [ "params.position_s" ], "params" : { "expr" :
"parseFloat(params_position_s) < 3" } }
```

unless

A logical function where processing will continue only if this function evaluates to false.

This function takes one additional property, 'expr', which is a JavaScript expression that must evaluate to a Boolean true/false. This property takes the same objects as the 'expr' function, described above. If this property is missing, the function will evaluate to false when any sourceField or targetField is present.

Example:

```
{ "type" : "unless", "sourceFields" : [ "params.position_s" ], "params" : { "expr"
: "parseFloat(params_position_s) > 1" } }
```

Scripting Functions

script

A scripted function. Scripts are evaluated as snippets, not as a function, and are expected to operate directly on the source event and the result. Their final values are discarded, since snippets in JavaScript are treated as expressions that evaluate to a specific value.

This function ignores the sourceFields and targetFields properties. Instead, the snippets are passed the following properties:

- startScript: the script defined is executed when the aggregation for the next unique tuple starts.
- aggregateScript: the script defined is executed for each source event.
- finishScript: the script is defined when all events for the current tuple have been processed and the result is about to be returned.

Example:

```
{ "type" : "script", "sourceFields" : [ ], "params" : { "aggregateScript" :
"result.addField('script_event_id_ss', event.getFieldValue('id'));" } }, { "type" :
"script", "sourceFields" : [ ], "params" : { "aggregateScript" :
"event.addField('position_rnd_1', event.getFieldValue('params.position_s') + 1.0 -
Math.random());event.addField('position_rnd_2',
event.getFieldValue('params.position_s') + 5.0 - Math.random() * 10.0);" } }
```

expr

A script expression. The script is evaluated as a snippet, and its final value is assigned to the targetField.

This function has only one additional property, 'expr', which contains the script expression.

Example:

```
{ "type" : "expr", "sourceFields" : [ "query_s", "filters_s" ], "targetField" : "expr_s", "params" : { "expr" : "v = ''; if (value != null) v = value + ' '; v + query_s + '_' + filters_s" } }, { "type" : "expr", "sourceFields" : [ "params.position_s" ], "targetField" : "expr_d", "params" : { "expr" : "v = 0; if (value != null) v = parseFloat(value); v + parseFloat(params_position_s)" } }
```

Special Functions

noop

A function that does nothing (is non-operational). This is a fallback function when invalid function parameters or execution errors are encountered.

Example:

```
{ "type" : "noop" }
```

Built-in SQL Aggregation Jobs

In addition to information about built-in SQL aggregation jobs, this topic describes how to use the Catalog API to set up a project and data asset that can be used as input for SQL aggregation jobs.

SQL aggregation jobs can also be set up using source files in Cloud Storage buckets. For more information, see *Built-in SQL Aggregation Jobs Using Cloud Storage Buckets*.

When you create a new Fusion app, Fusion automatically creates the `COLLECTION_NAME_signals` and `COLLECTION_NAME_signals_aggr` collections, plus the aggregation jobs described below. You can view these jobs at **collections** > **jobs**.

Job	Default input collection	Default output collection	Default schedule
`COLLECTION_NAME_click_signals_aggregation`	`COLLECTION_NAME_signals`	`COLLECTION_NAME_signals_aggr`	Every 15 minutes
`COLLECTION_NAME_session_rollup`	`COLLECTION_NAME_signals`	`COLLECTION_NAME_signals`	Every 15 minutes

When you *enable recommendations*, Fusion automatically creates these additional aggregation jobs:

Job	Default input collection	Default output collection	Default schedule
`COLLECTION_NAME_user_item_prefs_agg`	`COLLECTION_NAME_signals`	`COLLECTION_NAME_recs_aggr`	Once per day
`COLLECTION_NAME_user_query_history_agg`	`COLLECTION_NAME_signals`	`COLLECTION_NAME_signals_aggr`	Once per day

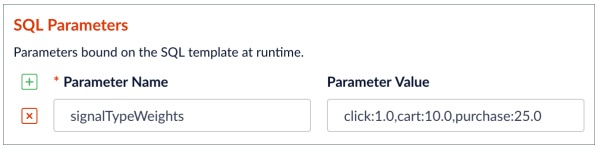
COLLECTION_NAME_click_signals_aggregation

The `COLLECTION_NAME_click_signals_aggregation` job computes a time-decayed weight for each document, query, and filters group in the signals collection. Fusion computes the weight for each group using an exponential time-decay on signal count (30 day half-life) and a weighted sum based on the signal type. This approach gives more weight to a signal that represents a user purchasing an item than to a user just clicking on an item.

	query	count_i	type	timstamp_tdt	user_id	doc_id	session_id	fusion_query_id
Required signals fields:								[*1*]

1 Required if you are using response signals.

You can customize the signal types and weights for this job by changing the `signalTypeWeights` SQL parameter in the Fusion Admin UI.



When the SQL aggregation job runs, Fusion translates the `signalTypeWeights` parameter into a `WHERE IN` clause to filter signals by the specified types (click, cart, purchase), and also passes the parameter into the `weighted_sum` SQL function. Notice that Fusion only displays the SQL parameters and not the actual SQL for this job. This is to simplify the configuration because, in most

cases, you only need to change the parameters and not worry about the actual SQL. However, if you need to change the SQL for this job, you can edit it under the **Advanced** toggle on the form.

A user can configure the `COLLECTION_NAME_click_signals_aggregation` job to use a parquet file as the source of raw signals instead of a signal Fusion collection.

1. Use *catalog api* to set up a "catalog project" in Fusion:

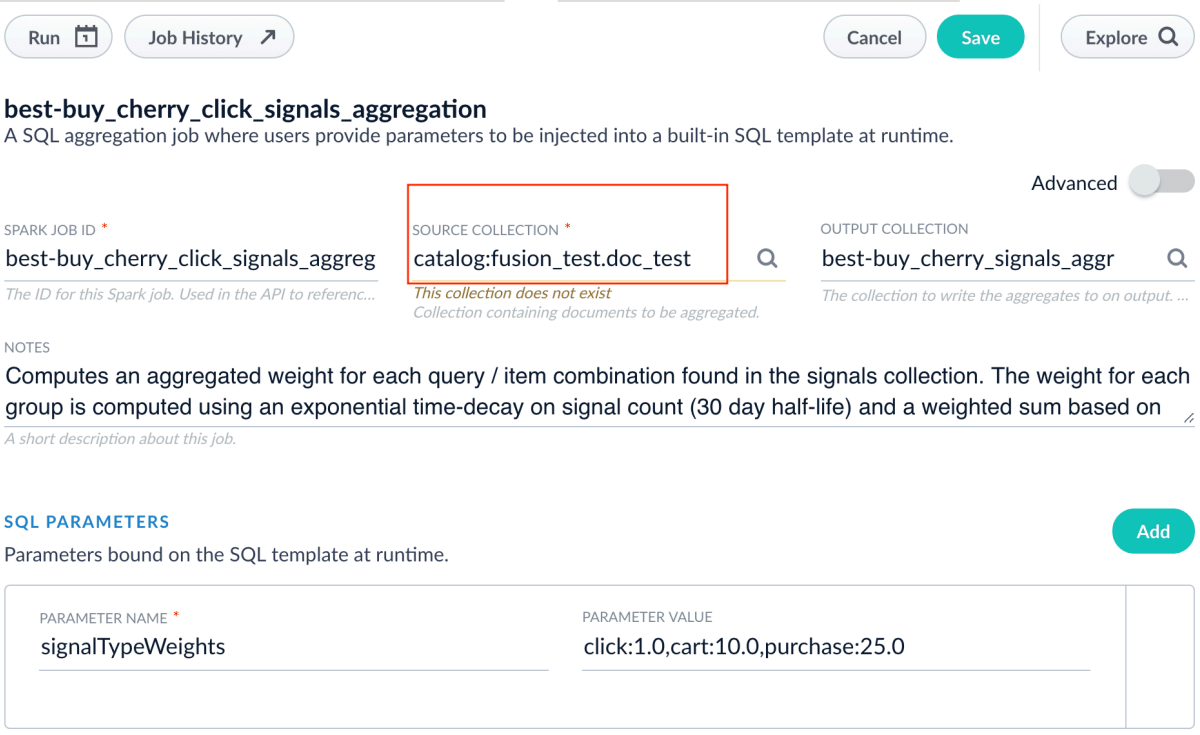
```
sample code: curl -u <username>:<pw> -X POST -H "Content-type:application/json"
--data-binary '{ "name": "fusion_test", "assetType": "project", "description":
"test", "cacheOnLoad": false }' http://FUSION_HOST:FUSION_PORT/api/catalog
```

2. Create an assets table in the project created in previous step:

```
sample code: curl -u <username>:<pw> -X POST -H "Content-type:application/json"
--data-binary '{ "name": "doc_test", "assetType": "table", "projectId":
"fusion_test", "description": "for documentation", "tags": ["fusion"], "format":
"parquet", "cacheOnLoad": false, "options" : [ "path -> <path to your .parquet
file>"] }' http://FUSION_HOST:FUSION_PORT/api/catalog/fusion_test/assets
```

The parquet file listed above needs to have all the fields in the SQL script which the `COLLECTION_NAME_click_signals_aggregation` job is selecting/using.

3. In the `COLLECTION_NAME_click_signals_aggregation` job, change "source" from `${collection}_signals` to `catalog:${project_name}.${asset_name}` (e.g. `catalog:fusion_test.doc_test` per the sample code).



4. Start the job.

COLLECTION_NAME_session_rollup

The `COLLECTION_NAME_session_rollup` job aggregates related user activity into a session signal that contains activity count, duration, and keywords (based on user search terms).

The Fusion App Insights application uses this job to show reports about user sessions.

Use the `elapsedSecsSinceLastActivity` and `elapsedSecsSinceSessionStart` parameters to determine when a user session is considered to be complete. You can edit the SQL using the **Advanced** toggle.

The `COLLECTION_NAME_session_rollup` job uses signals as the input collection and output collection. Unlike other aggregation jobs that write aggregated documents to the `COLLECTION_NAME_signals_aggr` collection, the `COLLECTION_NAME_session_rollup` job creates session signals and saves them to the `COLLECTION_NAME_signals` collection.

COLLECTION_NAME_user_item_prefs_agg

The `COLLECTION_NAME_user_item_preferences_aggregation` job computes an aggregated weight for each user/item combination found in the signals collection. The weight for each group is computed using an exponential time-decay on signal count (30 day half-life) and a weighted sum based on the signal type.

	query	count_i	type	timstamp_tdt	user_id	doc_id	session_id	fusion_query_id
Required signals fields:		✓	✓	✓	✓	✓		

This job is a prerequisite for the *BPR Recommender job*.

Job configuration tips:

- In the job configuration panel, click **Advanced** to see all of the available options.
- When aggregating signals for the first time, uncheck the **Aggregate and Merge with Existing** checkbox. In production, once the jobs are running automatically then this box can be checked. Note that if you want to discard older signals then by unchecking this box those old signals will essentially be replaced completely by the new ones.
- If the original signal data has missing fields, edit the SQL query to fill in missing values for fields such as “count_i” (the number of times a user interacted with an item in a session).
- Sometimes the aggregation job can run faster by unchecking the **Job Skip Check Enabled** box. Do this when first loading the signals.
- Use the `signalTypeWeights` SQL parameter to set the correct signal types and weights for your dataset. Its value is a comma-delimited list of signal types and their stakeholder-defined level of importance. Think of this numeric value as a weight that tells which type of signal is most important for determining a user’s interest in an item. An example of how to weight the signal types is shown below:

  ```
  signal_type_1:1.0, signal_type_2: 3.0, signal_type_3: 20.0
  ```

  Rank your signal types to determine which types should be added. Add only the signal types that are significant. Signal types that are not added to the list will not be included in the aggregation job, and for some signal types this is fine.

  The weights should be within orders of magnitude of each other. The spread of values should not be wide. For instance, `click:1.0, cart:100000.0` is too wide of a spread. The values of `click:1.0` and `cart:50.0` would be a reasonable setting, indicating that the signal type of `cart` is 50 times more important for measuring a user’s interest in an item.
- The Time Range field value is used in a weight decay function that reduces the importance of signals the older they are. This time range is in days and the default is 30 days. If you want to increase this time because the time duration of your signals is greater than 30 days, edit the SQL query to reflect the desired number of days. The SQL query is visible when you click **Advanced** in the job configuration panel. Modify the following line in the SQL query, changing "30 days" to your desired timeframe:

```
time_decay(count_i, timestamp_tdt, "30 days", ref_time, weight_d) AS
typed_weight_d
```

If recommendations are enabled for your collection, then the ALS recommender job is automatically created with the name `COLLECTION_NAME_item_recommendations` and scheduled to run after this job completes. Consequently, you should only run this aggregation once or twice a day, because training a recommender model is a complex, long-running job that requires significant resources from your Fusion cluster.

COLLECTION_NAME_user_query_history_agg

The `COLLECTION_NAME_user_query_history_aggregation` job computes an aggregated weight for each user/query combination found in the signals collection. The weight for each group is computed using an exponential time-decay on signal count (30 day half-life) and a weighted sum based on the signal type. Use the `signalTypeWeights` parameter to set the correct signal types and weights for your dataset. You can use the results of this job to boost queries for a user based on their past query activity.

	query	**count_i**	**type**	**timstamp_tdt**	**user_id**	**doc_id**	**session_id**	**fusion_query_id**
Required signals fields:	✓	✓	✓	✓	✓			

SQL Aggregation Examples

To acquaint you with how to use Spark SQL in SQL aggregations, here are several examples:

Perform the default SQL aggregation

This is the default SQL aggregation of signals for a base collection named `products`. It produces the same results as legacy aggregation:

```
SELECT SUM(count_i) AS aggr_count_i, query AS query_s, doc_id AS doc_id_s,
time_decay(count_i, date) AS weight_d FROM products_signals GROUP BY query, doc_id
```

Notice the following about this SQL:

- `SELECT SUM(count_i) AS aggr_count_i`. `count_i` is summed as `aggr_count_i`.
- `time_decay(count_i, date) AS weight_d`. The `time_decay` function computes the aggregated `weight_d` field. This function is a Spark UserDefinedAggregateFunction *(https://spark.apache.org/docs/latest/api/scala/index.html#org.apache.spark.sql.expressions.UserDefinedAggregateFunction)* (UDAF) that is built into Fusion. The function computes a weight for each aggregation group, using the count and an exponential decay on the signal timestamp, with a 30-day half life.
- `GROUP BY query, doc_id`. The GROUP BY clause defines the fields used to compute aggregate metrics, which are typically the `query`, `doc_id`, and any filters. With SQL, you have more options to compute aggregated metrics without having to write custom JavaScript functions (which would be needed to supplement legacy aggregations). You can also use standard WHERE clause semantics, for example, `WHERE type_s = 'add'`, to provide fine-grained filters.
- The `time_decay` function uses an abbreviated function signature, `time_decay(count_i, timestamp_tdt)`, instead of the full function signature shown in *Use Different Weights Based on Signal Types*.

An example of how SQL aggregation works
This is an example of how this aggregation works. Consider the following four input signals for a fictitious query `q1` and document `1`:

```
[{ "type_s":"add", "doc_id_s":"1", "query_s":"q1", "count_i":1,
"timestamp_tdt":"2017-07-11T00:00:00Z"}, { "type_s":"view", "doc_id_s":"1",
"query_s":"q1", "count_i":1, "timestamp_tdt":"2017-07-11T00:00:00Z"}, {
"type_s":"add", "doc_id_s":"1", "query_s":"q1", "count_i":1, "timestamp_tdt":"2017-
07-11T00:00:00Z"}, { "type_s":"view", "doc_id_s":"1", "query_s":"q1", "count_i":1,
"timestamp_tdt":"2017-07-11T00:00:00Z"}]
```

Fusion generates the following aggregated document for `q1`:

```
{ "aggr_count_i":4, "query_s":"q1", "doc_id_s":"1", "weight_d":0.36644220285922535,
"aggr_id_s":"products_sql_agg", "aggr_job_id_s":"15d4279d128T755e5137",
"flag_s":"aggr", "query_t":"q1", "aggr_type_s":"sql", "timestamp_tdt":"2017-07-
14T19:01:05.950Z"}
```

Use different weights based on signal types

This is a slightly more complex example that uses a subquery to compute a custom weight for each signal based on the signal type (`add` vs. `click`):

```
SELECT SUM(count_i) AS aggr_count_i, query_s, doc_id_s, time_decay(count_i,
timestamp_tdt, "5 days", ref_time, signal_weight) AS weight_d FROM (SELECT count_i,
query_s, doc_id_s, timestamp_tdt, ref_time, CASE WHEN type_s='add' THEN 0.25 ELSE
0.1 END AS signal_weight FROM products_signals) GROUP BY query_s, doc_id_s
```

Compute metrics for sessions

This aggregation query uses a number of Spark SQL functions *(https://spark.apache.org/docs/latest/api/scala/index.html#org.apache.spark.sql.functions$)* to compute some metrics for sessions:

```
SELECT concat_ws('||', clientip, session_id) as id, first(clientip) as clientip,
min(ts) as session_start, max(ts) as session_end, (unix_timestamp(max(ts)) -
unix_timestamp(min(ts))) as session_len_secs_l, sum(asInt(bytes)) as total_bytes_l,
count(*) as total_requests_l FROM sessions GROUP BY clientip, session_id
```

SQL Aggregation Jobs

A Spark SQL aggregation job where user-defined parameters are injected into a built-in SQL template at runtime.

Configuration properties

BPR Recommender Jobs

Use this job when you want to compute user recommendations or item similarities using a Bayesian Personalized Ranking (BPR) recommender algorithm.

The ALS recommender job is deprecated. Use this recommender job instead.

Default job name	`COLLECTION_NAME_bpr_item_recs`
Input	*Aggregated signals* (the `COLLECTION_NAME_recs_aggr` collection by default)
Output	• *Items-for-item recommendations* (the `COLLECTION_NAME_bpr_item_recs` collection by default) • *Items-for-user recommendations* (the `COLLECTION_NAME_bpr_user_recs` collection by default)

	query	count_i	type	timstamp_tdt	user_id	doc_id	session_id	fusion_query_id
Required signals fields:		✓	✓	✓	✓	✓		

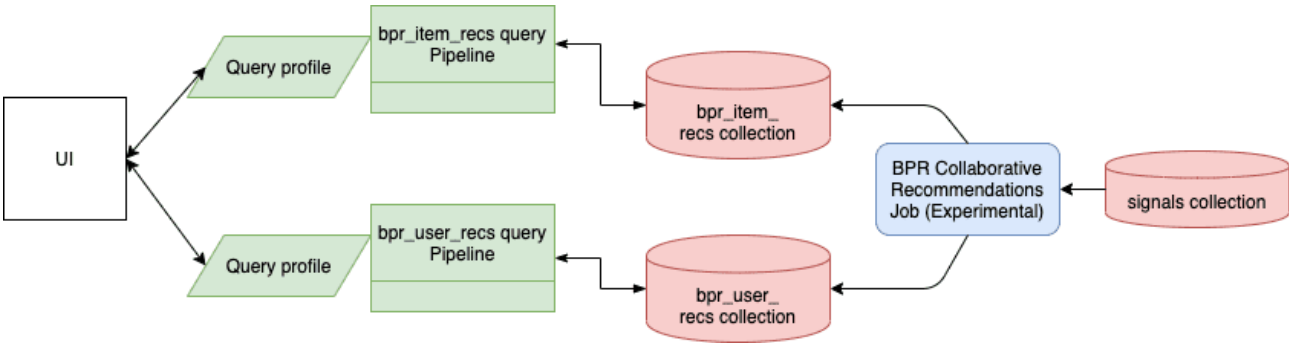
This job assumes that your signals collection contains the preferences of many users. It uses this collection of preferences to predict another user's preference for an item that the user has not yet seen:

- **User.** Use **Training Collection User Id Field** to specify the name of the user ID field, usually `user_id_s`.
- **Item.** Use **Training Collection Item Id Field** to specify the name of the item ID field, usually `item_id_s`.
- **Interaction-value.** Use **Training Collection Counts/Weights Field** to specify the name of the interaction value field, usually `aggr_count_i`.

Compared to *ALS-based recommenders*, BPR-based recommenders compare a pair of recommendations for a user instead of static `0, 1` input-based recommendations as in ALS.

You can also configure this job to read from or write to cloud storage. See *Configure An Argo-Based Job to Access GCS* and *Configure An Argo-Based Job to Access S3*.

BPR collaborative recommendations dataflow



If using Solr as the training data source, ensure that the source collection contains the `random_*` dynamic field defined in its managed-schema. This field is required for sampling the data. If it is not present, add the following entry to the managed-schema alongside other dynamic fields `<dynamicField name="random_*" type="random"/>` and <fieldType class="solr.RandomSortField" indexed="true" name="random"/> alongside other field types.

Tuning tips

The BPR Recommender job has a few unique tuning parameters compared to the ALS Recommender job:

- **Training Data Filtered By Popular Items**
  By setting the minimum number of user interactions required for items to be included in training and recommendations, you can suppress items that do not yet have enough signals data for meaningful recommendations.
- **Filter already clicked items**

This feature produces only "fresh" recommendations, by omitting items the user has already clicked. (It also increases the job's running time.)

- **Perform approximate nearest neighbor search**
  This option reduces the job's running time significantly, with a small decrease in accuracy. If your training dataset is very small, then you can disable this option.
- **Evaluate on test data**
  This feature samples the original dataset to evaluate how well the trained model predicts unseen user interactions. The clicks that are sampled for testing are not used for training. For example, with the default configuration, users who have at least three total clicks are selected for testing. For each of those users, one click is used for testing and the rest are used for training. The trained model is applied to the test data, and the evaluation results are written to the log.
- **Metadata fields for item-item evaluation**
  These fields are used during evaluation to determine whether pairs belong to the same category.

Query pipeline setup

- For *items-for-item recommendations*, *download the* `APPName_item_item_rec_pipelines_bpr.json` file and *import it* to create the query pipeline that consumes this job's output. See *Fetch Items-for-Item Recommendations (Collaborative/BPR Method)* for details.
- For *items-for-user recommendations*, *download the* `APPName_item_user_rec_pipelines_bpr.json` file and *import it* to create the query pipeline that consumes this job's output. See *Fetch Items-for-User Recommendations (Collaborative/BPR Method)* for details.

Configuration properties

Build Training Data Jobs

Use this job to build training data for query classification by joining signals data with catalog data. The output of this job can be used as input for the *Classification job*.

For detailed configuration steps, see *Classify New Queries*.

Configuration properties

Classification Jobs

This job analyzes how your existing documents are categorized and produces a classification model that can be used to predict the categories of new documents at index time.

For detailed configuration instructions and examples, see Classify New Documents at Index Time or Classify New Queries.

This job takes raw text and an associated single class as input. Although it trains on single classes, there is an option to predict the top several classes with their scores.

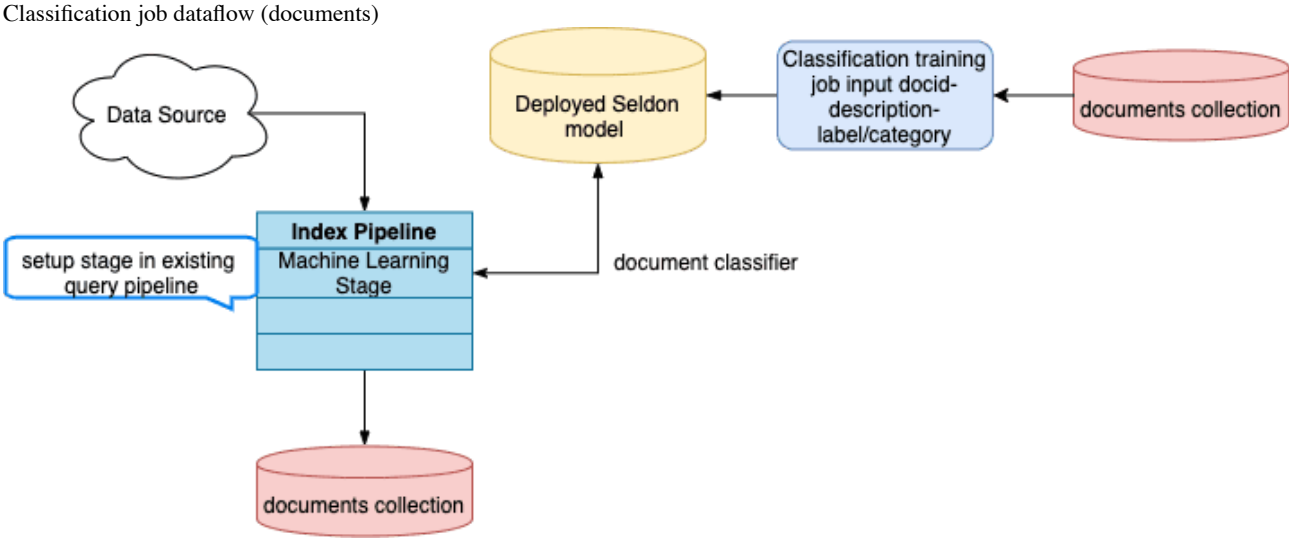
At a minimum, you must configure these:

- An ID for this job
- A **Method**; Logistic Regression is the default
- A **Model Deployment Name**
- The **Training Collection**
- The **Training collection content field**, the document field containing the raw text
- The **Training collection class field** containing the classes, labels, or other category data for the text

Classification at index time

Used at index time, a classification model can be applied to predict the categories of new, incoming documents. To train a model for this use case, use your main content collection as the training collection. The model requires at least 100 examples in the training data for each category predicted.

Classification job dataflow (documents)



Once you have run the job, you can specify the model name in the *Machine Learning Index Stage*.

Job flow

The first part of the job is *vectorization* which is the same for all available classification algorithms. Mainly it supports two types of featurization:

- **Character-based** - for queries or short texts, like document titles, sentences, and so on.
- **Word-based** - for long texts like paragraphs, documents, and so on.

The second part is *classification algorithms*:

- **Logistic Regression.** A classical algorithm with a good trade-off between training speed and results quality. It provides a robust baseline out of the box. Consider using it as a first choice.
- **StarSpace.** A deep learning algorithm that jointly trains to maximize similarity between text and correct class and minimize similarity between text and incorrect classes. This usually requires more tuning and time for training, but with potentially more accurate results. Consider using it and then tuning it if better results are needed.

The third part of the job deploys the new classification model to Fusion using Seldon Core.

Best practices

These tips describe how to tune the options under **Vectorization Parameters** for best results with different use cases.

Query intent / short texts

If you want to train a model to predict query intents or to do short text classification, then enable **Use Characters**.

Another vectorization parameter that can improve model quality is **Max Ngram size**, with reasonable defaults between 3 and 5.

The more character ngrams are used the bigger the vocabulary, so it is worthwhile to tune the **Maximum Vocab Size** parameter that controls how many unique tokens will be used. Lower values will make training faster and will prevent overfitting but might provide lower quality too. It's important to find a good balance.

Activating the advanced **Sublinear TF** option usually helps if characters are used.

Documents / long texts

If you want to train a model to predict classes for documents or long texts like one or more paragraphs, then uncheck **Use Characters**.

The reasonable values for word-based **Max Ngram size** are 2-3. Be sure to tune **Maximum Vocab Size** parameter too. Usually it's better to leave the advanced **Sublinear TF** option deactivated.

Performance tuning

If the text is very long and **Use Characters** is checked, the job may take a lot of memory and possibly fail if the amount of memory requested by the job is not available. This may result in pods being evicted or failing with OOM errors. If you see this happening, try the following:

- Uncheck **Use Characters**.
- Reduce the vocabulary size and ngram range of the documents.
- Allocate more memory to the pod.

Algorithm-specific

If you are going to train a model via LogisticRegression algorithm, dimensionality reduction usually doesn't help so it makes sense to leave **Reduce Dimensionality** unchecked. But scaling seems to improve results, so it's suggested to activate **Scale Features**.

For models trained by StarSpace algorithm it's vice-versa. Dimensionality reduction usually helps to get better results as well as much faster model training. But scaling usually doesn't help or might make results a little bit worse.

Index pipeline configuration

Model input transformation script

```
/* Name of the document field to feed into the model. */ var documentFeatureField =
"body_t" /* Model input construction. */ var modelInput = new java.util.HashMap()
modelInput.put("text", doc.getFirstFieldValue(documentFeatureField)) modelInput
```

Model output transformation script

```
var top1ClassField = "top_1_class_s" var top1ScoreField = "top_1_score_d"
doc.addField(top1ClassField, modelOutput.get("top_1_class")[0])
doc.addField(top1ScoreField, modelOutput.get("top_1_score")[0])
```

```
// In case if top_k_predictions are needed var top1ClassField = "top_1_class_s" var top1ScoreField = "top_1_score_d" var topKClassesField = "top_k_classes_ss" var topKScoresField = "top_k_scores_ds" var jsonOutput = JSON.parse(modelOutput.get("_rawJsonResponse")) var parsedOutput = {}; for (var i=0; i<jsonOutput["names"].length;i++){ parsedOutput[jsonOutput["names"][i]] = jsonOutput["ndarray"][i] } doc.addField(top1ClassField, parsedOutput["top_1_class"][0]) doc.addField(top1ScoreField, parsedOutput["top_1_score"][0]) if ("top_k_classes" in parsedOutput) { doc.addField(topKClassesField, new java.util.ArrayList(parsedOutput["top_k_classes"][0])) doc.addField(topKScoresField, new java.util.ArrayList(parsedOutput["top_k_scores"][0])) }
```

Query pipeline configuration

Model input transformation script

```
var modelInput = new java.util.HashMap() modelInput.put("text", request.getFirstParam("q")) modelInput
```

Model output transformation script

```
// To put into request request.putSingleParam("class", modelOutput.get("top_1_class")[0]) request.putSingleParam("score", modelOutput.get("top_1_score")[0]) // Or for example to apply Filter Query request.putSingleParam("fq", "class:" + modelOutput.get("top_1_class")[0])
```

```
// To put into query context context.put("class", modelOutput.get("top_1_class")[0]) context.put("score", modelOutput.get("top_1_score")[0])
```

```
// To put into response documents (can be done only after Solr Query stage) var docs = response.get().getInnerResponse().getDocuments(); var ndocs = new java.util.ArrayList(); for (var i=0; i<docs.length;i++){ var doc = docs[i]; doc.putField("query_class", modelOutput.get("top_1_class")[0]) doc.putField("query_score", modelOutput.get("top_1_score")[0]) ndocs.add(doc); } response.get().getInnerResponse().updateDocuments(ndocs);
```

```
// In case if top_k_predictions are needed // To put into response documents (can be done only after Solr Query stage) var jsonOutput = JSON.parse(modelOutput.get("_rawJsonResponse")) var parsedOutput = {}; for (var i=0; i<jsonOutput["names"].length;i++){ parsedOutput[jsonOutput["names"][i]] = jsonOutput["ndarray"][i] } var docs = response.get().getInnerResponse().getDocuments(); var ndocs = new java.util.ArrayList(); for (var i=0; i<docs.length;i++){ var doc = docs[i]; doc.putField("top_1_class", parsedOutput["top_1_class"][0]) doc.putField("top_1_score", parsedOutput["top_1_score"][0]) if ("top_k_classes" in parsedOutput) { doc.putField("top_k_classes", new java.util.ArrayList(parsedOutput["top_k_classes"][0])) doc.putField("top_k_scores", new java.util.ArrayList(parsedOutput["top_k_scores"][0])) } ndocs.add(doc); } response.get().getInnerResponse().updateDocuments(ndocs);
```

Configuration properties

Cluster Labeling Jobs

Use this job when you already have clusters or well-defined document categories, and you want to discover and attach keywords to see representative words within those existing clusters. (If you want to create new clusters, use the *Document Clustering job*.)

Configuration properties

Content-Based Recommender Jobs (Experimental)

Use this job when you want to compute item similarities based on their content, such as product descriptions.

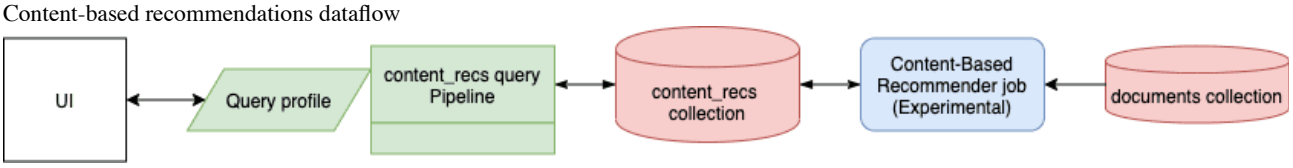
Default job name	`COLLECTION_NAME_content_recs`
Input	Searchable content from the primary collection.
Output	*Items-for-item recommendations* (the `COLLECTION_NAME_content_recs` collection by default)

First, item content is vectorized; different vectorization methods are available. Then, similar items are selected based on cosine similarity ("nearest neighbor") between their vectors.

At a minimum, you must specify these:

- An ID for this job
- The name of the training collection, that is, the collection with your content
- An output collection; create a separate collection for this
- The name of the ID field for documents in the training collection, such as `item_id_s`
- The names of one or more content fields in the training collection

Content-based recommendations dataflow



Tuning tips

- Configure **Metadata fields for item-item evaluation** to use those fields during evaluation to determine whether pairs belong to the same category.
- **Perform approximate nearest neighbor search** is enabled by default to significantly reduce the job's running time, with a small decrease in accuracy. If your training dataset is very small, then you can disable this option.
- If your content contains a lot of domain-specific jargon, enable **Use Word2Vec for vectorization**.
- If your documents are too short or too long, enable **Use TF-IDF for vectorization**.

Query pipeline setup

Download the `APPName_item_item_rec_pipelines_content.json` file and *import it* to create the query pipeline that consumes this job's output. See *Fetch Content-Based Items-for-Item Recommendations* for details.

Configuration properties

Create Collections in Milvus Jobs

Creates collections with specified parameters in *Milvus*.

Configuration properties

Create Indexes in Milvus Jobs

Creates indexes for specified collections in *Milvus*.

Configuration properties

Create Seldon Core Model Deployment

Use this job to deploy a Seldon Core Model into the Fusion cluster.

For information to deploy available Fusion Seldon models, see the:

- *sentiment-general model* trained on short sentences and text for intent prediction. For more information, see *Sentiment analysis and prediction*.
- *sentiment-reviews model* trained on customer reviews and longer text. For more information, see *Sentiment analysis and prediction*.
- *pretrained scispacy model* trained on biomedical text.
- *qna-coldstart-large model* trained on a variety of corpuses and tasks. For more information, see *Smart answers*.
- *qna-coldstart-mulitlingual model* trained on a variety of corpuses and tasks that supports 16 languages. For more information, see *Smart answers*.

Configuration properties

Custom Python Jobs

The Custom Python job provides user the ability to run Python code via Fusion. This job supports Python 3.6+ code.

Usage

Python code can be entered directly in the job configuration editor or you can reference a script that has been uploaded to the blob store. Additional Python libraries or files can be supplied via a Python files configuration.

Examples

Example Python script that indexes data from parquet to Solr via a Fusion index pipeline:

```
from pyspark.sql import SparkSession import sys """ Python script that indexes data
from parquet to Fusion via index pipeline zkhost of the cluster is always passed as
the first argument """ if __name__ == "__main__": if len(sys.argv) != 5:
print("Program requires 3 arguments. Args passed {}. Add <parquet_file>
COLLECTION_NAME <index-pipeline> via submit args in the job
config".format(sys.argv), file=sys.stderr) sys.exit(-1) zkhost = sys.argv[1]
parquet_file = sys.argv[2] collection = sys.argv[3] index_pipeline = sys.argv[4]
sparkSession = SparkSession.builder\ .appName("load_data_to_index_pipeline")\
.config("fs.gs.impl", "com.google.cloud.hadoop.fs.gcs.GoogleHadoopFileSystem")\
.config("fs.AbstractFileSystem.gs.impl",
"com.google.cloud.hadoop.fs.gcs.GoogleHadoopFS")\ .getOrCreate()
df=sparkSession.read.parquet(parquet_file).limit(1000)
df.write.format("lucidworks.fusion.index").option( "zkhost", zkhost).option(
"collection", collection).option( "pipeline", index_pipeline).save()
```

The above script is wrapped in the `script` variable of the job config and several arguments are passed via the `submitArgs` configuration key:

```
{ "type": "custom_python_job", "id": "test_python_script", "script": "\nfrom
pyspark.sql import SparkSession\n\nimport sys\n\n\"\"\"\nPython script that indexes
data from parquet to Fusion via index pipeline\n \nzkhost of the cluster is always
passed as the first argument\n\"\"\"\nif __name__ == \"__main__\":\n if
len(sys.argv) != 5:\n print(\"Program requires 3 arguments. Args passed {}. Add
<parquet_file> COLLECTION_NAME <index-pipeline> via submit args in the job
config\".format(sys.argv), file=sys.stderr)\n sys.exit(-1)\n\n zkhost =
sys.argv[1]\n parquet_file = sys.argv[2]\n collection = sys.argv[3]\n
index_pipeline = sys.argv[4] \n\n sparkSession = SparkSession.builder
.appName(\"load_data_to_index_pipeline\") .config(\"fs.gs.impl\",
\"com.google.cloud.hadoop.fs.gcs.GoogleHadoopFileSystem\")
.config(\"fs.AbstractFileSystem.gs.impl\",
\"com.google.cloud.hadoop.fs.gcs.GoogleHadoopFS\") .getOrCreate()\n\n
df=sparkSession.read.parquet(parquet_file).limit(1000)\n
df.write.format(\"lucidworks.fusion.index\").option(\n \"zkhost\",
zkhost).option(\n \"collection\", collection).option(\n \"pipeline\",
index_pipeline).save()\n", "submitArgs": [ "gs://smartdata-
datasets/best_buy_product_catalog.snappy.parquet", "demo", "demo" ],
"verboseReporting": true }
```

For more PySpark script examples, see https://github.com/apache/spark/blob/v2.4.4/examples/src/main/python *(https://github.com/apache/spark/blob/v2.4.4/examples/src/main/python)*.

Configuration

Apache Arrow is installed to the image and the two settings below are enabled by default. If you want to disable arrow optimization, set these properties to false in the job config or in job-launcher config map:

```
spark.sql.execution.arrow.enabled true spark.sql.execution.arrow.fallback.enabled
true
```

See also https://spark.apache.org/docs/latest/sql-pyspark-pandas-with-arrow.html *(https://spark.apache.org/docs/latest/sql-pyspark-pandas-with-arrow.html)*.

Available libraries

These libraries are available in the Fusion Spark image:

- `numpy`
- `scipy`
- `matplotlib`
- `pandas`
- `scikit-learn`

Adding libraries

If you need to add extra libraries to run your code, you can upload the Python egg files to the blob store and reference their blob IDs in the job configuration.

However, machine learning libraries (like `tensorflow`, `keras`, and `pytorch`) are not easy to install with that approach. To install those libraries, follow this approach instead:

1. Use this example `Dockerfile` to extend from the base image:

```
FROM lucidworks/fusion-spark:5.0.2 RUN pip3 install tensorflow keras pytorch
```

2. Build the Docker image and publish it to your own Docker registry.
3. Once the image is built, the custom image can be specified in the Spark settings via `spark.kubernetes.driver.container.image` and `spark.kubernetes.executor.container.image`.

Configuration properties

Custom Spark Jobs

Run a custom Spark job.

This job is available in Fusion 5.4.2 and later.

Fusion identifies all JARs in the blob store and appends them to the Spark environment. Make sure to upload the JAR file to the *blob store* with `resourceType=spark:jar` so your custom Spark job will see it.

Data Augmentation Job

Use this job to augment training and/or testing data for use with other jobs, such as Smart Answer, Classification, Recommender.

This job takes in data specified by the user, performs one or more of the specified augmentation tasks, and writes the output back to Solr or Cloud.

The Benefits of Augmentation Tasks

The augmentation tasks can improve the models trained on it by adding the augmented data back into the model, thereby increasing the quantity of training data when there isn't enough. They can also allow you to test the robustness of the models by training them on the source data, then testing them on augmented data. Both introduce variation that will make the models better equipped to handle different types of text. For more details on this process, see Data Augmentation *(https://en.wikipedia.org/wiki/Data_augmentation)*.

The amount of extra augmented data generated will depend on the task and the parameters used. In an ideal scenario with one task applied to one field and little to no record filtering, you can expect to double the amount of the original data.

Task Description

Each task supports a variety of languages. Refer to the description of each task for details.

- **Backtranslation**
  Translates the input data into one or more intermediate languages before translating it back to the source language. The process introduces changes in the syntax and grammar of the input text without changing the semantics. Because this task uses a deep learning model, Facebook's M2M-100 *(https://arxiv.org/pdf/2010.11125.pdf)*, to perform translations, a GPU is recommended for fast processing.

  If the backtranslation is of poor quality, try increasing the beam size. However, this will consume more memory and take more time. You could also try changing the intermediate languages to use languages that are similar to each other. For example, if your source language is Korean, translating to Chinese and/or Japanese and back might give you better results than translating to Spanish.

  Use the synonym substitution job as an alternative if you're unable to provision the necessary hardware and/or this job is taking too long. Note that the synonym substitution job does not support the same languages.

  Supported Languages: Chinese, Dutch, English, French, German, Hebrew, Italian, Japanese, Korean, Polish, Spanish, Ukrainian

Backtranslation with Korean text may result in errors if run on GKE with Kubernetes master version v1.16.15-gke.4901, Kernel version: 4.19.112+, and Container runtime version: docker://19.3.1 on Google's Container Optimized OS for Docker. To resolve, upgrade to a higher version of K8s master, kernel, and container runtime.

- **Synonym Substitution**
  Takes in the input text and substitutes some words with synonyms derived from the included wordner/ppdb dictionaries or user-supplied dictionaries. The user-supplied dictionaries must be submitted in the lucene/solr synonym format as shown in the example below.

  Example **synonyms.txt** file:

```
#some test synonym mappings unlikely to appear in real input text aaa => aaaa
bbb => bbbb1 bbbb2 ccc => cccc1,cccc2 a\=>a => b\=>b a\,a => b\,b
fooaaa,baraaa,bazaaa # Some synonym groups specific to this example
GB,gib,gigabyte,gigabytes MB,mib,megabyte,megabytes Television, Televisions, TV,
TVs #notice we use "gib" instead of "GiB" so any WordDelimiterFilter coming
#after us won't split it into two words. # Synonym mappings can be used for
spelling correction too pixima => pixma
```

  Supported Languages: Chinese, Dutch, English, French, German, Hebrew, Italian, Japanese, Polish, Spanish

Boosted synonyms are not supported. This synonym mapping file should be uploaded to the blob store, and a blob store path should then be passed to the job.

- **Keystroke Misspelling**
  Simulates typos one might make based on the layout of the keyboard. For example, if typing in English on a QWERTY keyboard layout, they might accidentally replace the "y" with a "t" while typing the word "keyboard" because "y" and "t" are next to each other on the keyboard. Currently, only QWERTY keyboard layouts are supported.

  The user can provide their own keyboard mapping as a JSON file uploaded to the fusion blob store. The JSON file should be in the following format: `{"a":"x", "b":"v", …}`.

Supported Languages: Dutch, English, French, German, Hebrew, Italian, Polish, Spanish, Ukrainian

- **Split word**
  Randomly splits words by introducing a space `“ “` at some random point in the word.
  Supported Languages: Dutch, English, French, German, Italian, Polish, Spanish

Delete Collections in Milvus Jobs

Deletes specified collections in *Milvus*.

Configuration properties

Delete Indexes in Milvus Jobs

Deletes specified indexes in *Milvus*.

Configuration properties

Delete Seldon Core Model Deployment Job

Use this job to remove a Seldon Core deployment from the Fusion cluster.

Configuration properties

Document Clustering Jobs

The Document Clustering job uses an unsupervised machine learning algorithm to group documents into clusters based on similarities in their content. You can enable more efficient document exploration by using these clusters as facets, high-level summaries or themes, or to recommend other documents from the same cluster. The job can automatically group similar documents in all kinds of content, such as clinical trials, legal documents, book reviews, blogs, scientific papers, and products.

<table>
<tr><td>Input</td><td>Searchable content (your primary collection)</td></tr>
<tr><td>Output</td><td>The job adds the following fields to the content documents:
<table>
<tr><td><code>cluster_id</code></td><td>The IDs associated with each cluster so that we can easily identify the clusters by number. A negative number means the document is an outlier or extremely long.</td></tr>
<tr><td><code>cluster_label</code>, <code>cluster_label_txt</code></td><td>Unique keywords assigned to each cluster so that there are no overlapping keywords between the clusters.
The cluster label <code>long_doc</code> is applied to very lengthy documents.
The cluster label <code>short_doc</code> is applied to very short documents.
Outliers are grouped with cluster labels like <code>outlier_group0</code>, <code>outlier_group1</code>, and so on.</td></tr>
<tr><td><code>dist_to_center</code></td><td>The document’s distance from its corresponding cluster center. The shorter the distance, the closer the document is to the center of the cluster.</td></tr>
<tr><td><code>freq_terms</code>, <code>freq_terms_txt</code></td><td>The most frequent words in the cluster.</td></tr>
<tr><td><code>clustering_model_id</code></td><td>The ID of the Document Clustering job that attached these fields.</td></tr>
</table></td></tr>
</table>

The Document Clustering job is an end-to-end job that includes the following:

- Document preprocessing
- Separating out extremely lengthy documents and outliers (de-noise)
- Automatic selection of the number of clusters
- Extracting cluster keyword labels

You can choose between multiple clustering and featurization methods to find the best combination of methods.

Cluster labels and frequent terms

Cluster labels are the terms that best represent the documents in a given cluster. They are typically words that can be found in the documents closest to the cluster centroid, so the `dist_to_center` field can be used to sort documents by similarity to the `cluster_label` field.

Frequent terms are the terms that appear most frequently in documents in a given cluster. Different clusters may have overlapping frequent terms. Some of the frequent terms may also appear in the cluster label.

Configuration tips

The minimum required fields to configure are straightforward:

- **Training Collection**
- **Output Collection**
- **Field to Vectorize**

When you first create a new Document Clustering job, set the **Training Collection** to point to a special collection that contains a sample set of documents. Set the **Output Collection** to a new, empty collection. That way, you can test the job quickly over a smaller input collection and you can clear the output collection after each test.

	When you create your sample collection and your test output collection, uncheck **Enable Signals** to prevent secondary collections from being created.

When you are satisfied with the results of the job, set *both* the **Training Collection** and the **Output Collection** to your primary collection. It may take some time for the job to run over your entire primary collection. At the end, the new clustering fields are added to your existing searchable content.

The sections below discuss additional ways to tune your configuration for the best results.

Use stopwords

The quality of your cluster labels depends on the quality of your *stopwords*. Fusion comes with a very basic stopword list, but there may be other words in your corpus that you want to remove. If you see terms in your cluster labels that your do not want, add them to your stopwords and then re-run the job.

Scale your Spark resources

Make sure that you have enough resources in Spark to run the job, especially if you have many documents or your documents are long. If Spark doesn't have enough memory, you may see out-of-memory errors in the Job History tab.

If there are obstacles to scalability, you can index a subset of the text from each document then run the job on this downsized version of the documents. The algorithm does not need all of the text to generate meaningful clusters. Stopword removal also decreases the document size significantly.

Select the clustering method

The job provides these clustering methods:

- Hierarchical Bisecting Kmeans ("hierarchical")
  The default choice is "hierarchical," which is a mixed method between Kmeans and hierarchical clustering. It can tackle the problem of uneven cluster sizes produced by standard Kmeans, and is more robust regarding initialization. In addition, it runs much faster than the standard hierarchical-clustering method, and has fewer problems dealing with overlapping topic documents.
- Standard Kmeans ("kmeans")
  For use cases such as novel and review clustering, several words can express similar meanings. In that case, Kmeans can perform well in combination with the Word2Vec featurization method described below. This method is also helpful when you have a corpus with a large vocabulary. Kmeans also works well when clusters are "convex", meaning that they are regularly shaped.

There are two ways to configure the number of clusters:

- Setting **Number Of Clusters** speeds up the processing time, but finding the best single value can be difficult unless you know exactly how many clusters are in the dataset.
- Setting **Minimum Possible Number Of Clusters** and **Maximum Possible Number Of Clusters**) allows Fusion to test up to 20 different values within the configured range to find the best number of clusters for your dataset based on metrics like how far each datapoint is from its associated cluster center. This optimizes the algorithm to detect true groups of similar documents and thus create better-quality clusters.
  For example, if `kMin=2` and `kMax=100`, then the job searches through 2, 7, 12, …, 100 with a step size of 5. A large `kMax` can increase the running time. The algorithm incurs a penalty if k is unnecessarily large. You can use the parameter `kDiscount` to reduce this penalty and use a larger k-chosen. However, if `kMax` is small (for example, 10 or less), then not using a discount (`kDiscount=1`) is recommended.

Select the featurization method

The job provides two text-vectorization methods:

- TFIDF
  You can trim out noisy terms for TFIDF by specifying the **Min Doc Support** and **Max Doc Support** parameters (minimum and maximum number of documents that contain the term).
  If you are using the hierarchical clustering method, then you should apply the TFIDF featurization method; it can provide better-detailed clusters for use cases like clustering email or product descriptions.
- Word2Vec
  Word2Vec can reduce dimensions and extract contextual information by putting co-occurring words in the same subspace. However, it can also lose some detailed information by abstraction. If you assign **Word2Vec Dimension** an integer greater than 0, then Fusion chooses the Word2Vec method over TFIDF.
  If you are using the standard kmeans clustering method, then you should enable the Word2Vec featurization method.
  For a large corpus dataset with a big vocabulary, Word2Vec is preferred to help deal with the dimensionality.

Configure de-noise parameters

The job provides three layers of protection from the impact of noisy documents:

- In the `analyzerConfig` parameter, you can specify stopword deletion, stemming, short token treatment, and regular expressions. The `analyzerConfig` is used in the featurization step.
- You can add an optional phase to separate out documents that are extremely long or short (as measured by the number of tokens). Extremely short or long documents can contaminate the clustering process. Documents with a length between **Length Threshold for Short Doc** and **Length Threshold for Long Doc** are kept for clustering.
- The job performs outlier detection using the Kmeans method. Fusion groups documents into **Number of Outlier Groups**, then trims out clusters with a size less than **Outlier Cutoff** as outliers.
  Outliers are documents that are too distant from the nearest centroid or nearest neighbors. These distant outliers are grouped into outlier clusters and labeled as `outlier_group0`, `outlier_group1` … `outlier_groupN`, where N is the value of **Number of Outlier Groups**.

Configure cluster labelling

The configuration parameter **Number Of Keywords For Each Cluster** determines the number of keywords to pick to describe each cluster. Test different values until you find that the cluster labels give an accurate depiction of the data contained within a given cluster.

Evaluating and tuning the results

When the job has finished:

1. Navigate to the output collection.
2. Open the Query Workbench.
3. Click **Add a Field Facet** and select `cluster_label`.
   The `cluster_label` field contains the five terms that best define the center of each cluster.
4. Click **Add a Field Facet** again and select `freq_terms`.
   The `freq_terms` field contains the five terms that appear most often in each cluster. In many cases, the frequent terms in a cluster are also among those that best define it.

	The `dist_to_center` field can be used to sort documents by similarity to their cluster labels.
5. Explore the facets to determine whether the clusters are useful.
   - Be sure to examine the `long_doc` cluster and any `outlier_group<n>` clusters. If no outliers are detected, consider increasing `outlierK` or `outlierThreshold`.
   - If you see terms that you do not want in your cluster labels, add them to your stopwords.
   - To tune the granularity of your clusters, adjust the values of **Min Possible Number of Clusters** and **Max Possible Number of Clusters**. Increasing these values will break up some of the bigger groups into smaller ones. Decreasing the values can consolidate smaller groups into larger ones. Experiment until you find the level of granularity that produces the most meaningful clusters.
   - If the clusters are very uneven, such as when most documents are in one large cluster:
     - Try increasing the outlier cutoff (some outlier groups are being labeled as clusters).
     - Try increasing k (you have many clusters combined into one).
   - If you have many outlier groups, but only a few docs per outlier group, try increasing the outlier cutoff to avoid saturating your number of outlier groups before all outliers have been removed.
   - If the corpus is large and the clustering job is taking too long:
     - Use Kmeans and Word2Vec (Kmeans does not usually do well with TF*IDF).
     - Use an exact k value (**Number Of Clusters**) instead of a range (**Min Possible Number of Clusters** and **Max Possible Number of Clusters**).
   - If the clustering job completes successfully according to logs, but is not writing data to Solr, check the schema specs on output fields. Solr strings have a maximum length that Spark strings do not. Use text type if your outputs are very long.

	this problem will only appear if reading from and writing to different collections.
6. Re-run the job and examine the facets again to see whether the results are more useful.

Back up your primary collection

Before running this job over your primary collection, make sure you have a backup of the original content. This can come in handy if you change your mind about the results and want to overwrite the document clustering fields.

Configuration properties

Download Blob

Download and update a blob.

Configuration properties

Ground Truth Jobs

Estimate ground truth queries using click signals and query signals, with document relevance per query determined using a click/skip formula.

> Pair this job with the *Ranking Metrics job* to calculate relevance metrics, such as nDCG.

Configuration properties

Head/Tail Analysis Jobs

Perform head/tail analysis of queries from collections of raw or aggregated signals, to identify underperforming queries and the reasons. This information is valuable for improving overall conversions, Solr configurations, auto-suggest, product catalogs, and SEO/SEM strategies, in order to improve conversion rates.

Default job name	`COLLECTION_NAME_head_tail`
Input	Raw or aggregated signals (the `COLLECTION_NAME_signals` or `COLLECTION_NAME_signals_aggr` collections by default)
Output	• Rewrites for underperforming queries (the `_query_rewrite_staging` collection by default) • Analytics tables (the `COLLECTION_NAME_job_reports` collection by default)

	query	count_i	type	timstamp_tdt	user_id	doc_id	session_id	fusion_query_id
Required signals fields:								

A minimum of 10,000 signals is required to successfully run this job.

You can review the output from this job by navigating to **Relevance** > **Rules** > **Rewrite** > **Head/Tail**. See *Underperforming query rewriting* for more information.

Head/tail analysis configuration

The job configuration must specify the following:

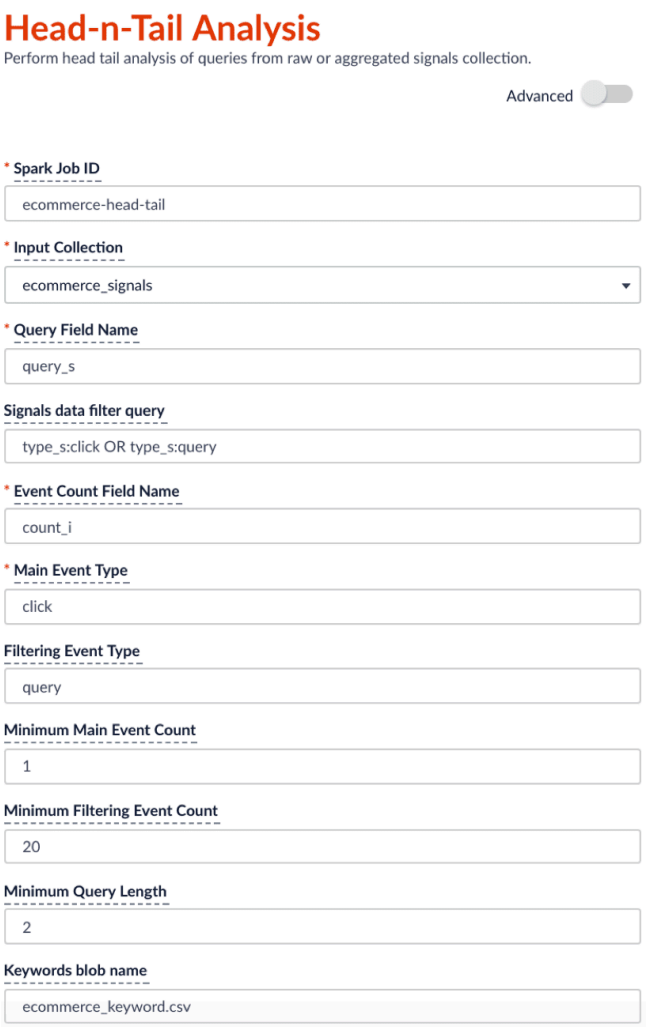
- The *signals* collection (the **Input Collection** parameter)
  Signals can be raw (the `COLLECTION_NAME_signals` collection) or *aggregated* (the `_signals_aggr` collection).
- The query string field (the **Query Field Name** parameter)
- The event count field
  For example, if signal data follows the default Fusion setup, then `count_i` is the field that records the count of raw signals and `aggr_count_i` is the field that records the count after aggregation.

The job allows you to analyze query performance based on two different events:

- The main event (the `mainType`/**Main Event Type** parameter)
- The filtering/secondary event (the `filterType`/**Filtering Event Type** parameter)
  If you only have one event type, leave this parameter empty.

For example, if you specify the main event to be clicks with minimum count of 0 and the filtering event to be queries with minimum count of 20, then the job will filter on the queries that get searched at least 20 times and check among those popular searched queries to see which ones didn't get clicked at all or only a few times.

An example configuration is shown below:



	The suggested schedule for this head-n-tail analysis job is to run bi-weekly or monthly. You can change schedule under the run panel.

Job output

By default, the output collection is the `<input-collection>_job_reports` collection. The head/tail job adds a set of analytics results tables to the collection. You can find these table names in the `doc_type_s` field of each document:

- `overall_distribution`
- `summary_stat`
- `queries_ordered`
- `tokens_ordered`
- `queryLength`
- `tail_reasons`
- `tail_rewriting`

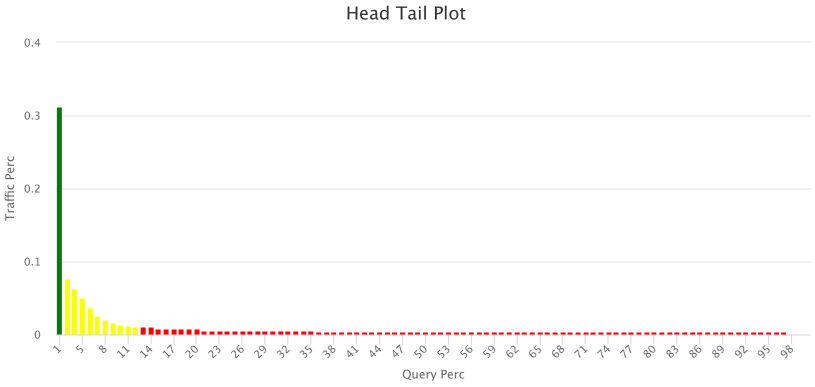
You can use *App Insights* to visualize each of these tables:

1. In the Fusion workspace, navigate to **Analytics** > **App Insights**.
   The App Insights dashboard appears.
2. On the left, click **Analytics** .
3. Under **Standard Reports**, click **Head Tail analysis**.
   The Head/Tail Analysis job output tables appear. These are described in more detail below.

Head/Tail Plot (`overall_distribution`)

This head/tail distribution plot provides an overview of the query traffic distribution. In order to provide better visualization, the unique queries are in descending order based on traffic and put into bins of 100 queries on the x axis, with the sum of traffic coming from each bin on the y axis.

For example, the head/tail distribution plot below shows a long tail, indicating that the majority of queries produce very little traffic. The goal of analyzing this data is to shorten that tail, so that a higher proportion of your queries produce traffic.



- Green = head
- Yellow = torso
- Red = tail

Summary Stats (`summary_stat`)

This user-configurable summary statistics table shows how much traffic is produced by various query groups, to help understand the head/tail distribution.

Summary Stats

Statistics	Value
total number of unique queries	22231
top 100 queries leads to ?% total events	21.24%
top 1.0% queries leads to ?% total events	31.06%
number of queries that constitute the top 75.0% of all events	3060
number of queries that constitute the top 50.0% of all events	680
number of queries that constitute the top 25.0% of all events	140
last 1.0% of total events spreaded in how many queries	1613
how many queries leads to zero events.	628
how many queries have less than 5 events	19734
events threshold for tail.	5
events threshold for head.	39

You can configure this table before running the job. Click **Advanced** in the Head/Tail Analysis job configuration panel, then tune these parameters:

- Top X% Head Query Event Count (`topQ`)
- Number of Queries that Constitute X% of Total Events (`trafficPerc`)
- Bottom X% Tail Query Event Count (`lastTraffic`)
- Event Count Computation Threshold (`trafficCount`)

Query Details (`queries_ordered`)

The Query Details table helps you discover which queries are the best performers and which are worst. You can filter results by issuing a search in the search bar. For example, search "segment_s:tail" to get tail queries or search "num_events_l:0" to get zero results queries. (Note: field names are listed in the "what is this" toolkit).

Query Details

Order ID	Query	Traffic (Click)	Traffic (Query)	Traffic Perc (Click)	Length	Num Tokens	Segment
1	ipad	860	862	00.873%	4	1	head
5	2622037 2127204 2127213 2121716 2138291	466	467	00.473%	39	5	head
6	tablet	435	437	00.442%	6	1	head
7	ipod	422	424	00.429%	4	1	head
8	beat	407	409	00.413%	4	1	head
9	touchpad	406	407	00.412%	8	1	head
10	kindle	332	334	00.337%	6	1	head
11	macbook	326	328	00.331%	7	1	head
12	iphone	320	322	00.325%	6	1	head
15	headphone	269	271	00.273%	9	1	head
17	projector	258	260	00.262%	9	1	head

Top Tokens (`tokens_ordered`)

The "Top Tokens" table lists the number of times each token shown in the queries.

Top Tokens

Tokens	Count
tv	675
the	613
samsung	609
sony	543
case	514
laptop	476
iphone	442
dvd	396
ipod	395
hp	376
car	359
wireless	356

Query Length (`queryLength`)

This table shows how users are querying your database. Are most people searching very long strings or very short strings? These distributions will give you insight into how to tune your search engine to be performant on the majority of queries.

Query Length

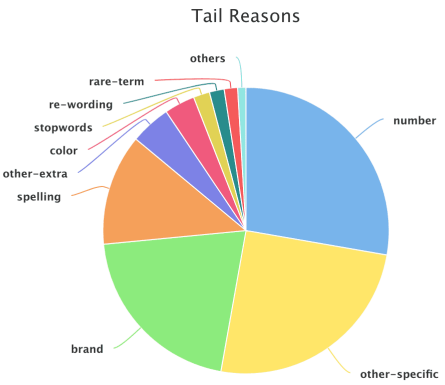
Avg String Length	One Word Perc	Two Words Perc	Three Words Perc	Four Words Perc	Five Words Perc	Six+ Words Perc
13	17.89	40.06	27.01	10.07	3.36	1.61

Tail Reasons table and pie chart (`tail_reasons`)

Based on the difference between the tail and head queries, the Head/Tail Analysis job assigns probable reasons for why any given query is a tail query. Tail reasons are displayed as both a table and a pie chart:

Tail Reasons

Count	Tail Reasons	Proportions
1249	number	6.61%
1129	other-specific	5.98%
930	brand	4.92%
565	spelling	2.99%
205	other-extra	1.08%
156	color	0.82%
83	stopwords	0.43%
75	re-wording	0.39%
68	rare-term	0.36%
42	others	0.22%



Pre-defined tail reasons

Based on Lucidworks' observations on different signal datasets, we summarize tail reasons into several pre-defined categories:

spelling	The query contains one or more misspellings; we can apply spelling suggestions based on the matching head.
number	The query contains an attribute search on a specific dimension. To normalize these queries we can parse the number to deal with different formatting, and/or pay attention to unit synonyms or enrich the product catalog. For example, "3x5" should be converted to "3’ X 5’" to match the dimension field.
other-specific	The query contains specific descriptive words plus a head query, which means the user is searching for a very specific product or has a specific requirement. We can boost on the specific part for better relevancy.
other-extra	This is similar to ‘other-specific’ but the descriptive part may lead to ambiguity, so it requires boosting the head query portion of the query instead of the specific or descriptive words.
rare-term	The user is searching for a rare item; use caution when boosting.
re-wording	The query contains a sequence of terms in a less-common order. Flipping the word order to a more common one can change a tail query to a head query, and allows for consistent boosting on the last term in many cases.
stopwords	Query contains stopwords plus head query. We would need to drop stopwords.

Custom dictionary

You can also specify your own attributes through a keywords file in CSV format. The header of the CSV file must be called "keyword" and "type", and stopwords must be called "stopword" for the program to recognize them.

Below is an example dictionary that defines "color" and "brand" reason types. The job will parse the tail query, assign reasons such as "color" or "brand", and perform filtering or focused search on these fields. (Note: color and brand are also the field names in your catalog.)

```
keyword,type a,stopword an,stopword and,stopword blue,color white,color black,color hp,brand samsung,brand sony,brand
```

How to install a custom dictionary

1. Construct the CSV file as described above.
2. *Upload the CSV file to the blob store*.
   Note the blob ID.
3. In the Head/Tail Analysis job configuration, enter the blob ID in the **Keywords blob name** (`keywordsBlobName`) field.

Head Tail Similarity (`tail_rewriting`)

For each tail query (the `tailQuery_orig` field), Fusion tries to find its closest matching head queries (the `headQuery_orig` field), then suggests a query rewrite (the `suggested_query` field) which would improve the query. The rewrite suggestions in this table can be implemented in a variety of ways, including utilizing rules editor or configuring a query parser that rewrites tail queries.

Tail Query	Suggested Rewriting	Matched Head	Reason Code	Matched Head (No Num)	Number	Other Specific	Number Unit	Tail Traffic
car amplifier	amplifier (car)^1.5	amplifier	other-specific	amplifier		car		5
headphone bluetooth	bluetooth headphone^2	bluetooth headphone	re-wording	bluetooth headphone				5
subwofer	subwoofer	subwoofer	spelling	subwoofer				5
lord of the ring blu ray	(lord of the ring)^2 blu ray	lord of the ring	other-extra	lord of the ring		blu ray		5
wireless n router	wireless router	wireless router	stopwords	wireless router				5
mw2	mw 2 mw2	mw	number	mw	2		mw2	5
hosa cable	cable hosa	cable	rare-term	cable		hosa		1
the lord of the ring blu ray	(lord of the ring)^2 the blu ray	lord of the ring	other-extra	lord of the ring		the blu ray		1

Configuration properties

Log Cleanup Jobs

Delete old log messages from system logs collection.

Configuration properties

Outlier Detection Jobs

Use this job when you want to find outliers from a set of documents and attach labels for each outlier group.

Configuration properties

Parallel Bulk Loader

The Parallel Bulk Loader (PBL) job enables bulk ingestion of structured and semi-structured data from big data systems, NoSQL databases, and common file formats like Parquet and Avro.

Configuration properties

Phrase Extraction Jobs

Identify multi-word phrases in signals.

Default job name	`COLLECTION_NAME_phrase_extraction`
Input	Raw signals (the `COLLECTION_NAME_signals` collection by default)
Output	Extracted phrases (the `COLLECTION_NAME_query_rewrite_staging` collection by default)

	query	count_i	type	timstamp_tdt	user_id	doc_id	session_id	fusion_query_id
Required signals fields:								

This job writes to the `COLLECTION_NAME_query_rewrite_staging` collection. It also uses reviewed documents from that collection to improve the accuracy of the job. You can review, edit, deploy, or delete output from this job using the *Query Rewriting*.

Fusion ships with the OpenNLP Maxent model *(http://opennlp.sourceforge.net/models-1.5/en-pos-maxent.bin)* already loaded in the *blob store*.

This job's output, and output from the *Token and Phrase Spell Correction job*, can be used as input for the *Synonym Detection job*.

Minimum configuration

For most use cases, the minimum configuration for this job consists of these fields:

- `id`/**Spark Job ID**
  Give this job an arbitrary ID string.
- `trainingCollection`/**Training Collection**
  Specify the input collection.
- `fieldToVectorize`/**Field to Vectorize**
  Specify the field in the input collection where phrases can be found.
- `outputCollection`/**Output Collection**
  Specify the collection in which the output documents should be indexed.

	When running this job over a content document collection, be sure to set `attachPhrases`/**Extract Key Phrases from Input Text** to "true". The default is "false", which works well when running the job over a signals collection.

Output documents

By default, the job only outputs the phrases found from the original document. In each row of the phrases output, these fields are most useful:

- The phrase itself is in the `phrases_s` field, which can be used for faceting.
- The `likelihood_d` field gives the likelihood that the phrase is legitimate, from 0 to infinity.
  Low-probability phrases are automatically trimmed from the results.
- When a phrase's likelihood value is ambiguous, the `review` field is set to "true" to indicate that the phrase should be reviewed.
- A `phrase_count` field indicates the number of instances of the phrase in the input collection.

The complete list of output fields is shown below.

Output fields

`aggr_id_s`	The name of the Phrase Extraction job that generated this document.
`doc_type_s`	This is always `key_phrases` for documents generated by a Phrase Extraction job.
`id`	A unique ID for this document.
`input_collection`	The collection used for this job's input.
`likelihood_d`	The likelihood that this `phrases_s` is a phrase, from 0 to infinity.
`phrase_count`	The number of occurrences of this phrase in the input collection.
`phrases_s`	The phrase detected by the job.
`review`	"True" indicates that this may not be a valid phrase and should be reviewed.
`score`	This is always "1".
`timestamp`	The date and time when the document was generated.
`word_num_i`	The number of words in this phrase.
`_version_`	An internal Solr field used for partial updates.

If the `attachPhrases`/**Extract Key Phrases from Input Text** parameter is set to "true", then the job also outputs the original documents from the input collection with an appended field, `phrases_extracted_tt`, that lists the extracted phrases from this document.

The way to distinguish the phrases output from the original document output is by the field `doc_type_s`, with one of these values:

- `key_phrases` denotes phrases output.
- `original_doc_with_phrases` denotes the original documents.

Configuration properties

Query-to-Query Collaborative Similarity Job

Train a collaborative filtering matrix decomposition recommender using SparkML's Alternating Least Squares (ALS) *(https://spark.apache.org/docs/2.1.0/ml-collaborative-filtering.html)* to batch-compute query-query similarities. This can be used for *items-for-query recommendations* as well as *queries-for-query recommendations*.

	query	count_i	type	timstamp_tdt	user_id	doc_id	session_id	fusion_query_id
Required signals fields:	✓	✓	✓	✓	✓			

Configuration properties

Query-to-Query Session-Based Similarity Jobs

This recommender is based on co-occurrence of queries in the context of clicked documents and sessions. It is useful when your data shows that users tend to search for similar items in a single search session. This method of generating query-to-query recommendations is faster and more reliable than the Query-to-Query Similarity recommender job, and is session-based unlike the similar queries previously generated as part of the *Synonym Detection job*.

Default job name	`COLLECTION_NAME_query_recs`
Input	Raw signals (the `_<collection_signals` collection by default).
Output	*Queries-for-query recommendations* (the `COLLECTION_NAME_queries_query_recs` collection by default)

	query	count_i	type	timstamp_tdt	user_id	doc_id	session_id	fusion_query_id
Required signals fields:	✓	✓	[*2*]	[*3*]	[*4*]	✓	✓	

2 Required if you want to weight types differently.

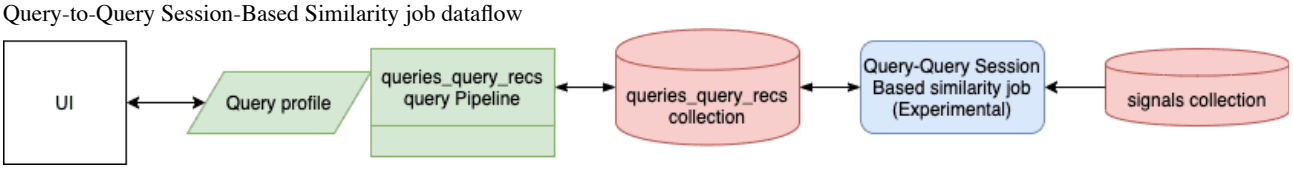
3 Required if you want to use time decay.

4 Required if no `session_id` field is available. Either `user_id` or `session_id` is needed on response signals if doing any click path analysis from signals.

The job generates two sets of recommendations based on the two approaches described below, then merges and de-duplicates them to present unique query-recommender pairs.

Similar queries based on documents clicked	Similar queries based on co-occurrence in sessions
Queries are considered for recommendation if two queries have similar sets of document IDs clicked according to the signals data. This is directly implemented from the similar queries portion of the *Synonym Detection job*. This approach can work on both raw and aggregated signals.	Queries are considered for recommendation if two queries have co-occurred in the same session based on the assumption that users search for similar items in a single search session (this may or may not hold true depending on the data). This approach, based on session/user IDs, needs raw signals to work.

Query-to-Query Session-Based Similarity job dataflow



A default Query-to-Query Session-Based Similarity job (`COLLECTION_NAME_query_recs`) and a dedicated *collection* and *pipeline* are created when you *enable recommendations for a collection*.

At a minimum, you must configure these:

- an ID for this job
- the input collection containing the signals data, usually `COLLECTION_NAME_signals`
- the data format, usually `solr`
- the query field name, usually `query_s`
- the document ID field name, usually `doc_id_s`
- optionally, the user session ID or user ID field name

If this field is not specified, then the job generates click-based recommendations only, without session-based recommendations.

Data tips

- Running the job on other types of data than signals is not recommended and may yield unexpected results.
- To get about 90% query coverage with the query pipeline, we recommend a raw signals dataset of about ~170k unique queries. More signals will generally improve coverage.
- On a raw signal dataset of about 3 million records, the job finishes execution in about 7-8 minutes on two executor pods with one CPU and about 3G of memory each. Your performance may vary depending on your configuration.

Boosting recommendations

Generally if a query and recommendation has some token overlap, then they're closely related and we want to highlight these. Therefore, query-recommendation pair similarity scores can be boosted based on token overlap. This overlap is calculated in terms of the number or fractions of tokens that overlap.

For example, consider the pair ("a red polo shirt", "red polo"). If the minimum match parameter is set to 1, then there should be 1 token in common. For this example there is 1 token in common and therefore it is boosted. If it is set to 0.5, then at least half of the tokens from the shorter string (in terms of space separated tokens) should match. Here, the shorter string is "red polo" which is 2 tokens long. Therefore, to satisfy the boosting requirement, at least 1 token should match.

Tuning tips

These tuning tips pertain to the advanced Model Tuning Parameters:

- **Special characters to be filtered out.** Special characters can cause problems with matching queries and are therefore removed in the job.
  Only the characters are removed, not the queries, so a query like `ps3$` becomes `ps3`.
- **Query similarity threshold.** This is for use by the similar queries portion of the job and is the same as that used in the Synonym and Similar Queries Detection job.
- **Boost on token overlap.** This enables or disables boosting of query recommendation pairs where all or some tokens match. How much match is required to boost can be configured using the next parameter.
  For example, if this is enabled, then a query-recommendation pair like `(playstation 3, playstation console)` is boosted with a similarity score of 1, provided the minimum match is set to 1 token or 0.5.
- **Minimum match for token overlap.** Similar to the `mm` param in Solr, this defines the number/fraction of tokens that should overlap if boosting is enabled. Queries and recommendations are split by " " (space) and each part is considered a token. If using a less-than sign (<), it must be escaped using a backslash.
  The value can be an integer, such as 1, in which case that many tokens should match. So in the previous example, pair is boosted because the term "playstation" is common to both and the `mm` value is set to 1.
  The value can also be a fraction, in which case that fraction of the shorter member of the query and recommendation pair should match. For example, if the value is set to 0.5 and query is 4 tokens long and recommendation is 6 tokens long then there should be at least 2 common tokens between query and recommendation.
  Here the stopwords specified in the *list of stopwords* are ignored while calculating the overlap.
- **Query clicks threshold.** The minimum number of clicked documents needed for comparing queries.
- **Minimum query-recommendation pair occurrence count.** Minimum limit for the number of times a query-recommendation pair needs to be generated to make it to the final similar query recommendation list. Default is set to 2. Higher value will improve quality but reduce coverage.

The similar queries collection

The following fields are stored in the `COLLECTION_NAME_queries_query_recs` collection:

- `query_t`
- `recommendation_t`
- `similarity_d`, the similarity score
- `source_s`, the approach that generated this pair, one of the following: `SessionBased` or `ClickedDocumentBased`
- `query_count_l`, the number of times the query occurred in signals
- `recommendation_count_l`, the number of times recommendations occurred in signals
- `pair_count_l`, the number of instances of the pair generated in the final recommendations using either of the approaches
- `type_s`, always set to `similar_queries`

The query pipeline

When you enable recommendations, a default query pipeline, `COLLECTION_NAME_queries_query_recs`. is created.

Configuration properties

REST Call Jobs

A versatile job type that runs an arbitrary REST/HTTP/Solr command.

REST jobs can call external systems using:

- any HTTP or HTTPS request: `<http-protocol>://<path>`; requires JWT header.
- a Solr request: `solr://COLLECTION_NAME/…`; authentication is handled.
- a Fusion service request: `extservice://SERVICE_NAME/<path>`; authentication is handled.

For example, you can configure this type of job to delete your old signals data:

1. Navigate to **Collections** > **Jobs**.
2. Click **Add** and select **REST Call**.
   The REST Call job configuration panel appears.
3. Enter an arbitrary ID for this job, such as "Delete-old-signals".
4. Enter the following endpoint URI, substituting the name of your signals collection for `signalsCollectionName`:

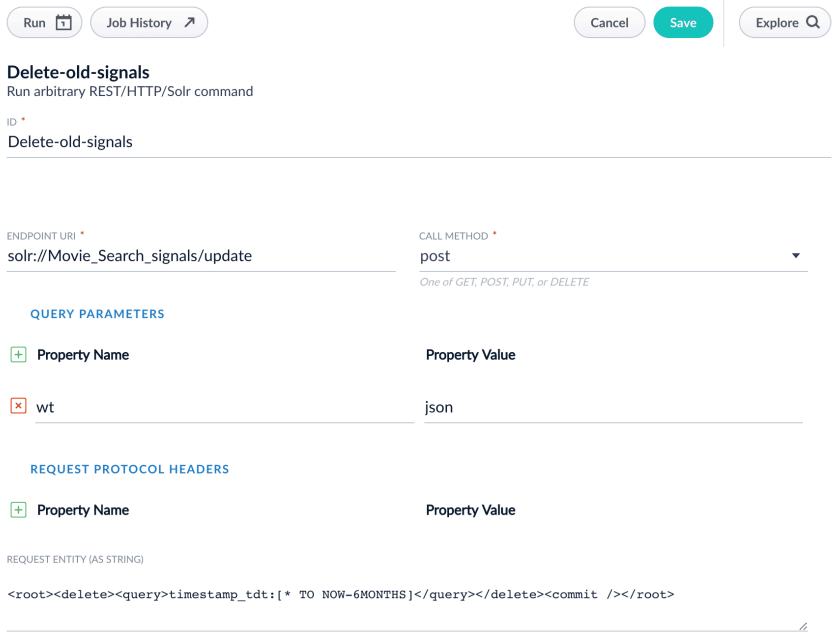
```
solr://signalsCollectionName/update
```

5. In the **Call Method** field, select "post".
6. Under **Query Parameters**, enter the property name "wt" with the property value "json".
7. In the **Request entity (as string)** field, enter the following:

```
<root><delete><query>timestamp_tdt:[* TO NOW-2WEEKS]</query></delete><commit />
</root>
```

See Working with Dates *(https://solr.apache.org/guide/8_6/working-with-dates.html)* for details about date formatting.

Your job configuration should look similar to this:



You can configure a schedule for this job at **System** > **Scheduler**.

Configuration properties

Ranking Metrics Jobs

Calculate relevance metrics (nDCG and so on) by replaying ground truth queries against catalog data using variants from an experiment.

Configuration properties

Recreate a Milvus Collection

Use this process to:

- Recreate a lost Milvus collection.
- Create a Milvus collection for a different field.
- Create a Milvus collection that uses a different encryption model.

Prerequisites

- Used the Smart Answers pipeline to index data, which created the Milvus and Solr collections.
- The created Solr collection still exists, which is used as the Source Collection.

Process

1. Create the Milvus collection. See *Create Collections in Milvus* for more information.

	If the Milvus collection still exists and you want to overwrite it, select the **Override Collections** checkbox. This option deletes the current collection, allowing you to create a new collection.

2. Edit the Smart Answers Index Pipeline as follows:
   - Disable the Solr Indexer stage.
   - If you are creating a Milvus collection for a different field, or one that uses a different encryption model, access the **Encode into Milvus** stage and:
     - Specify the **model**, the **Encoder Output Vector**, and the **Field to Encode**.
     - Verify the **Milvus Collection Name** refers to the new Milvus collection.
3. Navigate to the Datasources panel.
4. Select **Add a Solr connector** and enter the following information:
   - **Pipeline ID.** Smart Answers pipeline
   - **Solr Connection Type.** SolrCloud
   - **SolrCloud Zookeeper Host String.** Execute the following curl command. The response displays the SolrCloud Zookeeper Host String value. In the following `RESPONSE` example, the value you would set for the SolrCloud Zookeeper Host String is `ns-zookeeper-0.ns-zookeeper-headless:2181`.

     ```
     curl -u USERNAME:PASSWORD
     https://FUSION_HOST:FUSION_PORT/api/searchCluster/default
     ```

     RESPONSE

     ```
     { "id": "default", "connectString": "ns-zookeeper-0.ns-zookeeper-
     headless:2181", "zkClientTimeout": 30000, "zkConnectTimeout": 60000, "cloud":
     true, "bufferFlushInterval": 1000, "bufferSize": 100, "concurrency": 10,
     "authConfig": { "authType": "none" }, "validateCluster": true }
     ```

   - **Source Collection.** Original collection from the initial Smart Answers pipeline indexing job
5. Navigate to Index Workbench and load the new solr-index datasource.
6. Select **Start Job** to run the Solr datasource through the updated Smart Answers pipeline.
7. When the data is reindexed, edit the Smart Answers pipeline and re-enable the Solr Indexer stage.

SQL-Based Experiment Metric Jobs

This job is created by an experiment in order to calculate an objective.

This job is deprecated in Fusion 5.2.x.

	query	count_i	type	timstamp_tdt	user_id	doc_id	session_id	fusion_query_id
Required signals fields:								

Configuration properties

Script Jobs

Run a custom Scala script as a Fusion job.

Configuration properties

Smart Answers Coldstart Training

Train a *Smart Answers* model on a *cold start* (unsupervised) basis, with pre-trained or trained embeddings, and deploy the trained model to the ML Model Service.

See *Train A Smart Answers Cold Start Model* for configuration instructions.

Configuration properties

Smart Answers Evaluate Pipeline

Evaluate the performance of a *Smart Answers* pipeline.

See *Evaluate a Smart Answers Query Pipeline* for configuration instructions.

Configuration properties

Smart Answers Supervised Training

Train a *Smart Answers* model on a supervised basis, with pre-trained or trained embeddings, and deploy the trained model to the ML Model Service.

See *Train A Smart Answers FAQ Model* for configuration instructions.

Configuration properties

Synonym Detection Jobs

Use this job to generate pairs of synonyms and pairs of similar queries. Two words are considered potential synonyms when they are used in a similar context in similar queries.

Default job name	`COLLECTION_NAME_synonym_detection`
Input	• Aggregated signals (the `COLLECTION_NAME_signals_aggr` collection by default) • Spell Correction job output (the `COLLECTION_NAME_query_rewrite_staging` collection by default) • Phrase Extraction job output (the `COLLECTION_NAME_query_rewrite_staging` collection by default)
Output	Synonyms (the `COLLECTION_NAME_query_rewrite_staging` collection by default)

	query	count_i	type	timstamp_tdt	user_id	doc_id	session_id	fusion_query_id
Required signals fields:								

For best job speed and to avoid memory issues, use *aggregated signals* instead of raw signals as input for this job.

Output from the *Token and Phrase Spell Correction job* and the *Phrase Extraction job* can be used as input for this job.

To:

- Review, edit, deploy, or delete output from this job, see *Query Rewriting UI*
- Review, edit, or add synonyms, see *Use Synonym Detection*

Input

This job takes one or more of the following as input:

- *Signal data* (required)
- *Misspelling job results*
- *Phrase detection job results*
- *Keywords*
- *Custom synonyms*

Signal data

This input is required; additional input is optional. *Signal data* can be either raw or aggregated. The job runs faster using *aggregated signals*. When raw signals are used as input, this job performs the aggregation.

Use the `trainingCollection`/**Input Collection** parameter to specify the collection that contains the signal data.

Misspelling job results

Token and Phrase Spell Correction job results can be used to avoid finding mainly misspellings, or mixing synonyms with misspellings.

Use the `misspellingCollection`/**Misspelling Job Result Collection** parameter to specify the collection that contains these results.

Phrase detection job results

Phrase Extraction job results can be used to find synonyms with multiple tokens, such as "lithium ion" and "ion battery".

Use the `keyPhraseCollection`/**Phrase Extraction Job Result Collection** parameter to specify the collection that contains these results.

Keywords

A keywords list in the *blob store* can serve as a blacklist to prevent common attributes from being identified as potential synonyms.

The list can include common attributes such as color, brand, material, and so on. For example, by including color attributes you can prevent "red" and "blue" from being identified as synonyms due to their appearance in similar queries such as "red bike" and "blue bike".

The keywords file is in CSV format with two fields: `keyword` and `type`. You can add your custom keywords list here with the `type` value "stopwords". An example file is shown below:

```
keyword,type cu,stopword ft,stopword mil,stopword watt,stopword wat,stopword
foot,stopword feet,stopword gal,stopword unit,stopword lb,stopword wt,stopword
cc,stopword cm,stopword kg,stopword km,stopword oz,stopword nm,stopword qt,stopword
sale,stopword on sale,stopword for sale,stopword clearance,stopword gb,stopword
gig,stopword color,stopword blue,stopword white,stopword black,stopword
ivory,stopword grey,stopword brown,stopword silver,stopword light blue,stopword
light ivory,stopword light grey,stopword light brown,stopword light silver,stopword
light green,stopword
```

Use the `keywordsBlobName`/**Keywords Blob Store** parameter to specify the name of the blob that contains this list.

Custom Synonyms

For some deployments, there might be a need to use existing synonym definitions. You can import existing synonyms into the Synonym Detection Jobs as a text file. Upload your synonyms text file to the blob store and reference that file when creating the job.

Output

The output collection contains two tables distinguished by the `doc_type` field.

The similar queries table

If `query` leads to clicks on documents 1, 2, 3, and 4, and `similar_query` leads to clicks on documents 2, 3, 4, and 5, then there is sufficient overlap between the two queries to consider them similar.

A statistic is constructed to compute similarities based on overlap counts and query counts. The resulting table consists of documents whose `doc_type` value is "query_rewrite" and `type` value is "simq". The similar queries table contains similar query pairs with these fields:

`query`	The first half of the two-query pair.
`similar_query`	The second half of the two-query pair.
`similarity`	A score between 0 and 1 indicating how similar the two queries are. All `similarity` values are greater than or equal to the configured Query Similarity Threshold to ensure that only high-similarity queries are kept and used as input to find synonyms.
`query_count`	The number of clicks received by the `query_count` query. To save computation time, only queries with at least as many clicks as the configured Query Clicks Threshold parameter are kept and used as input to find synonyms.
`similar_query_count`	The number of clicks received by the `similar_query_count` query.

The synonyms table

The synonyms table consists of documents whose `doc_type` value is "query_rewrite" and `type` value is "synonym":

`surface_form`	The first half of the two-synonym pair.
`synonym`	The second half of the two-synonym pair.
`context`	If there are more than two words or phrases with the same meaning, such as "macbook, apple mac, mac", then this field shows the group to which this pair belongs.
`similarity`	A similarity score to measure confidence.
`count`	The number of different contexts in which this synonym pair appears. The bigger the number, the higher the quality of the pair.
`suggestion`	The algorithm automatically selects `context`, synonym words or phrases, or the `synonym_group`, and puts it in this field. Use this field as the field to review.
`category`	Whether the synonym is actually a misspelling.

Configuration properties

Token and Phrase Spell Correction Job

Detect misspellings in queries or documents using the numbers of occurrences of words and phrases.

Default job name	`COLLECTION_NAME_spell_correction`
Input	Raw signals (the `COLLECTION_NAME_signals` collection by default)
Output	Synonyms (the `COLLECTION_NAME_query_rewrite_staging` collection by default)

	query	count_i	type	timstamp_tdt	user_id	doc_id	session_id	fusion_query_id
Required signals fields:								

This job extracts tail tokens (one word) and phrases (two words) and finds similarly-spelled head tokens and phrases. For example, if two queries are spelled similarly, but one leads to a lot of traffic (head) and the other leads to a little or zero traffic (tail), then it is likely that the tail query is misspelled and the head query is its correction.

If several matching head tokens are found for each tail token, the job can pick the best correction using multiple configurable criteria.

You can review, edit, deploy, or delete output from this job using the *Query Rewriting UI*.

Misspelled terms are completely replaced by their corrected terms. If you want to expand the query to include all alternative terms, set the synonyms to bi-directional. See *Synonym Detection* for more information.

This job's output, and output from the *Phrase Extraction job*, can be used as input for the *Synonym Detection job*.

Solr treats spelling corrections as synonyms. See the blog post Multi-Word Synonyms: Solr Adds Query-Time Support *(https://lucidworks.com/post/multi-word-synonyms-solr-adds-query-time-support//)* for more details.

1. Create a job

Create a Token and Phrase Spell Correction job in the Jobs Manager.

How to create a new job

1. In the Fusion workspace, navigate to > **Jobs**.
2. Click **Add** and select the job type **Token and phrase spell correction**.
   The New Job Configuration panel appears.

2. Configure the job

Use the information in this section to configure the Token and Phrase Spell Correction job.

Required configuration

The configuration must specify:

- **Spark Job ID.** Used in the API to reference the job. Maximum 63 alphabetic characters, hyphen (-), and underscore (_).
- **INPUT COLLECTION.** The `trainingCollection` parameter that can contain signal data or non-signal data. For signal data, select **Input is Signal Data** (`signalDataIndicator`). Signals can be raw (from the `_signals` collection) aggregated (from the `_signals_aggr` collection).
- **INPUT FIELD.** The `fieldToVectorize` parameter.
- **COUNT FIELD**
  For example, if signal data follows the default Fusion setup, then `count_i` is the field that records the count of raw signals and `aggr_count_i` is the field that records the count after aggregation.

See *Configuration properties* for more information.

Event types

The spell correction job lets you analyze query performance based on two different events:

- The main event (the **Main Event Type**/`mainType` parameter)
- The filtering/secondary event (the **Filtering Event Type**/`filterType` parameter)
  If you only have one event type, leave this parameter empty.

For example, if you specify the main event type to be `click` with a minimum count of 0 and the filtering event type to be `query` with a minimum count of 20, then the job:

- Filters on the queries that get searched at least 20 times.
- Checks among those popular queries to determine which ones didn't get clicked at all, or were only clicked a few times.

Spell check documents

If you unselect the **Input is Signal Data** checkbox to indicate finding misspellings from content documents rather than signals, then you do not need to specify the following parameters:

- Count Field
- Main Event Field
- Filtering Event Type
- Field Name of Signal Type
- Minimum Main Event Count
- Minimum Filtering Event Count

Use a custom dictionary

You can upload a custom dictionary of terms that are specific to your data, and specify it using the **Dictionary Collection** (`dictionaryCollection`) and **Dictionary Field** (`dictionaryField`) parameters. For example, in an e-commerce use case, you can use the catalog terms as the custom dictionary by specifying the product catalog collection as the dictionary collection and the product description field as the dictionary field.

Example configuration

This is an example configuration:

Token and phrase spell correction

Use this job to compute token and phrase level spell correction which you can use in your synonym list.

Advanced

* **Spark Job ID**

spell-correction

The ID for this Spark job. Used in the API to reference this job. Allowed characters: a-z, A-Z, dash (-) and underscore (_)

INPUT/OUTPUT PARAMETERS

* **Input Collection**

E-commerce_signals

Collection containing search strings and event counts. Should ideally be the signals collection.If an aggregation colle...

Output Collection

E-commerce_signals_aggr

Collection to store misspelling and correction pairs. Defaults to the aggr collection.

Data filter query

type:click OR type:search

Solr query to additionally filter the search strings. Please let it be empty if 'type' field is unavailable in the input colle...

Input is Signal Data ☑

FIELD PARAMETERS

* **Input Field**

query

Field containing search strings.

Main Event Type

click

The main signal event type (e.g. click) that the job is based on if input is signal data. E.g., if main type is click, then h...

Filtering Event Type

search

The secondary event type (e.g. response) that can be used for filtering out rare searches.Note: In order to use this `r...

Count Field

count_i

Solr field containing query count

MODEL TUNING PARAMETERS

Minimum Main Event Count

1

Minimum number of main events (e.g. clicks after aggregation) necessary for the query to be considered. The job wi...

Minimum Filtering Event Count

10

Minimum number of filtering events (e.g. searches after aggregation) necessary for the query to be considered. The ...

Maximum Edit Distance

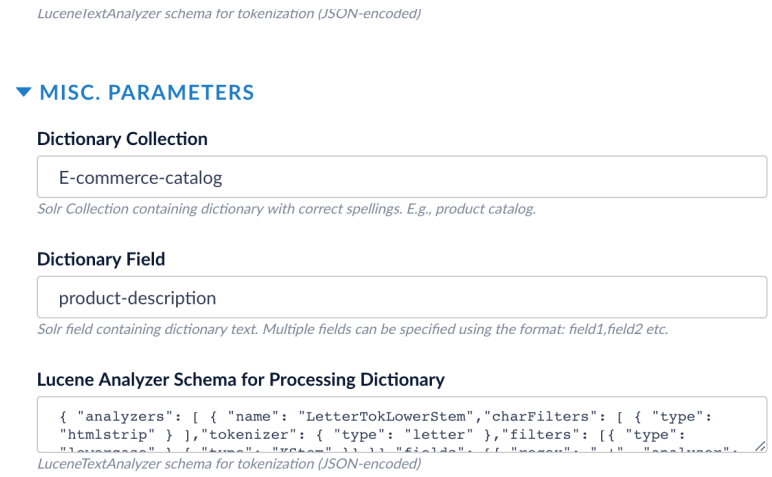
2

The minimum number of matches on starting characters. Note: Setting it to 0 may largely increase running time.

FEATURIZATION PARAMETERS

Lucene Analyzer Schema for Processing Queries

```
{ "analyzers": [ { "name": "LetterTokLowerStem","charFilters": [ { "type":
"htmlstrip" } ],"tokenizer": { "type": "letter" },"filters": [{ "type":
```



When you have configured the job, click **Save** to save the configuration.

3. Run the job

	If you are finding spelling corrections in aggregated data, you need to run an aggregation job before running the Token and Phrase Spelling Correction job. You *do not* need to run a Head/Tail Analysis job. The Token and Phrase Spell Correction job does the head/tail processing it requires.

How to run the job

1. In the Fusion workspace, navigate to > **Jobs**.
2. Select the job from the job list.
3. Click **Run**.
4. Click **Start**.

4. Analyze job output

After the job finishes, misspellings and corrections are output into the `query_rewrite_staging` collection by default; you can change this by setting the `outputCollection`.

An example record is as follows:

```
correction_s laptop battery mis_string_len_i 14 misspelling_s laptop baytery
aggr_job_id_s 162fcf94b20T3704c333 score 1 collation_check_s token correction
included corCount_misCount_ratio_d 2095 sound_match_b true id bf79c43b-fc6d-43a7-
931e-185fdac5b624 aggr_type_s tokenPhraseSpellCorrection aggr_id_s ecom_spell_check
correction_types_s phrase => phrase cor_count_i 68648960 suggested_correction_s
baytery=>battery cor_string_len_i 14 token_wise_correction_s baytery=>battery
cor_token_size_i 2 edit_dist_i 1 timestamp_tdt 2018-04-25T13:23:40.728Z mis_count_i
32768 lastChar_match_b true mis_token_size_i 2 token_corr_for_phrase_cnt_i 1
```

For easy evaluation, you can export the result output to a CSV file.

correction_s	mis_string_len_i	misspelling_s	aggr_job_id_s	score	collation_check_s	corCount_misCount_ratio_d	sound_match_b	id
frigidaire refrigerator	24	frigidaire refridgeratir	162fcf94b20T3704c333	1	token correction not included	52	false	00102e18-74ff-
laptop charger	13	laptop carger	162fcf94b20T3704c333	1	token correction not included	2708	true	00137869-f6e6-
logitech revue	15	logtitech revue	162fcf94b20T3704c333	1	token correction included	877	false	001bb1c8-008e
bryan	6	br yan	162fcf94b20T3704c333	1	N/A	120	true	001f1579-14e9
evo extend	10	evo extene	162fcf94b20T3704c333	1	token correction not included	62	true	00316b4a-0706
hakusho	7	hakisho	162fcf94b20T3704c333	1	shown in phrase list	93	true	0047b1e3-84d7
megami tensei	13	megami tensai	162fcf94b20T3704c333	1	token correction included	19	true	004b8f95-b003
dish network	12	dksh network	162fcf94b20T3704c333	1	token correction not included	31	true	004f816d-6f04-
tabet	5	tab t	162fcf94b20T3704c333	1	N/A	41	true	0067e70b-0879
laptop charger	14	laptip charger	162fcf94b20T3704c333	1	token correction not included	2708	true	0076e482-dd5e
beast	6	beatst	162fcf94b20T3704c333	1	not shown in phrase list	450	false	007e91a9-bd96
tab accessory	13	tab accestory	162fcf94b20T3704c333	1	token correction included	184	false	0092c3a5-0b62
cody simpson	12	cody simspon	162fcf94b20T3704c333	1	token correction not included	60	false	00b741ce-5efc
keyboard	9	keyboarrd	162fcf94b20T3704c333	1	not shown in phrase list	12682	true	00c464dd-f9e3
carter iv	10	ccarter iv	162fcf94b20T3704c333	1	token correction included	1353	true	00c5428f-b49b
elite receiver	13	elite reciver	162fcf94b20T3704c333	1	token correction not included	25	true	00c9f250-97eb
wireless headphone	18	wirelesa headphone	162fcf94b20T3704c333	1	token correction not included	2964	true	00d95133-1ce0
yamaha speaker	13	yamha speaker	162fcf94b20T3704c333	1	token correction included	40	true	00db05b1-afb2
joe bonamassa	13	joe bonamossa	162fcf94b20T3704c333	1	token correction not included	22	true	00dde26f-9ccc
ps headset	11	ps headseat	162fcf94b20T3704c333	1	token correction not included	1395	true	00e05075-c372
logitech g	11	logictech z	162fcf94b20T3704c333	1	token correction not included	303	false	00eafe9f-8653-
w magsafe	8	w magsfe	162fcf94b20T3704c333	1	token correction included	18	true	00fb9df0-b8cf-
gamera	7	gamerac	162fcf94b20T3704c333	1	not shown in phrase list	87	false	011b8aa2-d785
macintosh	8	mcintosh	162fcf94b20T3704c333	1	not shown in phrase list	92	true	0151d6b9-f0c4

5. Use spell correction results

You can use the resulting corrections in various ways. For example:

- Put misspellings into the synonym list to perform auto-correction.
- Help evaluate and guide the Solr spellcheck configuration.
- Put misspellings into typeahead or autosuggest lists.
- Perform document cleansing (for example, clean a product catalog or medical records) by mapping misspellings to corrections.

Useful output fields

In the job output, you generally only need to analyze the `suggested_corrections` field, which provides suggestions about using token correction or whole-phrase correction. If the confidence of the correction is not high, then the job labels the pair as "review" in this field. Pay special attention to the output records with the "review" labels.

With the output in a CSV file, you can sort by `mis_string_len` (descending) and `edit_dist` (ascending) to position more probable corrections at the top. You can also sort by the ratio of correction traffic over misspelling traffic (the `corCount_misCount_ratio` field) to only keep high-traffic boosting corrections.

For phrase misspellings, the misspelled tokens are separated out and put in the `token_wise_correction` field. If the associated token correction is already included in the one-word correction list, then the `collation_check` field is labeled as "token correction include." You can choose to drop those phrase misspellings to reduce duplications.

Fusion counts how many phrase corrections can be solved by the same token correction and puts the number into the `token_corr_for_phrase_cnt` field. For example, if both "outdoor surveillance" and "surveillance camera" can be solved by correcting "surveillance" to "surveillance", then this number is 2, which provides some confidence for dropping such phrase corrections and further confirms that correcting "surveillance" to "surveillance" is legitimate.

You might also see cases where the token-wise correction is not included in the list. For example, "xbow" to "xbox" is not included in the list because it can be dangerous to allow an edit distance of 1 in a word of length 4. But if multiple phrase corrections can be made by changing this token, then you can add this token correction to the list.

	Phrase corrections with a value of 1 for `token_corr_for_phrase_cnt` and with `collation_check` labeled as "token correction not included" could be potentially-problematic corrections.

Fusion labels misspellings due to misplaced whitespaces with "combine/break words" in the `correction_types` field. If there is a user-provided dictionary to check against, and both spellings are in the dictionary with and without whitespace in the middle, we can treat these pairs as bi-directional synonyms ("combine/break words (bi-direction)" in the `correction_types` field).

The `sound_match` and `lastChar_match` fields also provide useful information.

Job tuning

The job's default configuration is a conservative, designed for higher accuracy and lower output. To produce a higher volume of output, you can consider giving more permissive values to the parameters below. Likewise, give them more restrictive values if you are getting too many results with low accuracy.

When tuning these values, always test the new configuration in a non-production environment before deploying it in production.
`trainingDataFilterQuery`/**Data filter query**
See *Event types* above, then adjust this value to reflect the secondary event for your search application. To query all data, set this to `*:*`.
`minCountFilter`/**Minimum Filtering Event Count**
Lower this value to include less-frequent misspellings based on the data filter query.
`maxDistance`/**Maximum Edit Distance**
Raise this value to increase the number of potentially-related tokens and phrases detected.
`minMispellingLen`/**Minimum Length of Misspelling**
Lower this value to include shorter misspellings (which are harder to correct accurately).

Query rewrite jobs post-processing cleanup

To perform more extensive cleanup of query rewrites, complete the procedures in *Query Rewrite Jobs Post-processing Cleanup*.

Configuration properties

Transfer Collection to Cloud Job

The Transfer Collection to Cloud job allows you to migrate or copy your Solr collection to cloud storage.

Trending Recommender Jobs

The Trending Recommender job analyzes signals to measure customer engagement over time. Use this job to identify spikes in popularity for specific items or queries, then display those items to your users or analyze the trends for business purposes. You can configure any time window, such as daily, weekly, or monthly.

Input	*signals* (the `COLLECTION_NAME_signals` collection by default)
Output	Trending items or queries

	query	**count_i**	**type**	**timstamp_tdt**	**user_id**	**doc_id**	**session_id**	**fusion_query_id**
Required signals fields:	[*5*]							

5 Required when identifying trending queries instead of trending items.

For detailed steps to configure this job, see *Identify Trending Documents or Products*.

Configuration properties

Upload Model Parameters to Cloud Job

The Upload Model Parameters to Cloud job allows you to take a trained model and store its training to a blob in cloud storage, such as Google Cloud Storage (GCS), Amazon S3 (S3), or Azure Blob Storage.

See *Creating the Upload Model Parameters to Cloud Job* for more information on creating the job and deploying a model with it.

	With the release of Fusion 5.4, the ***Create Seldon Core Model Deployment*** job has been updated in conjunction so that it can pull down the saved model blobs and use them in deploying a new instance of the model, allowing for easier transference and versioning of trained models for rollback and testing.

Pipeline Stages

Index Pipeline stages are used to create and modify PipelineDocument objects.

Query Pipeline stages are used to modify Request objects and Response objects.

Pipeline Stage Properties

Fusion associates a pipeline stage definition with a unique ID and stores the definition so that stages can be reused across pipelines and applications. In addition to an ID, all stages have the following properties:

- **type** (required): an enumeration, one of the defined Fusion pipeline stage types, e.g., "index-logging". If the Fusion UI is used to define the stage, this property is filled in automatically.
- **label** (optional) : a string field with a maximum length of 255 characters. The label is displayed on the Fusion UI.
- **skip** (optional): a boolean value, If true, pipeline processing bypasses this stage altogether. The default is **false**.
- **condition** (optional): a JavaScript expression that evaluates to true (1) or false (0). If this condition evaluates to false, this stage is skipped. The default is **true**.

Pipeline Condition Expression

The JavaScript expression specified in the condition property of a pipeline stage has access the pipeline objects.

Index Pipeline Stage Condition Example

An index PipelineDocument has two available variables: "doc" and "ctx".

Check whether pipeline document contains a named field:

```
doc.hasField("acl_ss")
```

Query pipeline stage Condition Example

Process query based the query Request object:

```
request.hasParam("fusion-user-name") && request.getFirstParam("fusion-user-
name").equals("SuperUser"); !request.hasParam("isFusionPluginQuery")
```

The first condition checks that the request parameter "fusion-user-name" is present and has the value "SuperUser". The second condition checks that the request parameter "isFusionPluginQuery" is not present.

Query Pipeline Stages Reference

A query pipeline is made up of a series of query stages that process incoming search queries.

A pipeline stage definition associates a unique ID with a set of properties. These definitions are registered with the Fusion API service and stored in ZooKeeper for re-use across pipelines and search applications.

Fusion includes a number of specialized query stages as well as a JavaScript stage that allows advanced processing via a JavaScript program.

Use the *The Query Workbench* to configure stages in a query pipeline.

Active Directory Security Trimming Stage

An Active Directory Security Trimming query pipeline stage retrieves an Active Directory user's security identifiers to build a security filter. This restricts the documents in the query result to only those documents for which a user has access permissions. Security trimming is commonly used in business to authenticate between administrative users and normal users, or to limit the site access of website users according to a login/password.

This stage supports *asynchronous processing*.

Example Stage Setup

Active Directory Security Trimming query stage setup:

Input

```
curl -u USERNAME:PASSWORD -X POST -H 'Content-type: application/json' -d
'{"id":"ADSecurity", "type": "active-directory-security-trimming",
"server":"ldap://hostname:port","bindName":"ADuser@example.com","bindPassword":"log
in1"}' http://FUSION_HOST:FUSION_PORT/api/query-stages/instances
```

Output

```
{ "type" : "active-directory-security-trimming", "id" : "ADSecurity", "server" :
"ldap://hostname:port", "bindName" : "ADuser@example.com", "bindPassword" :
"login1", "enableCache" : true, "cacheSize" : 1000, "expirationTime" : 3600, "skip"
: false, "label" : "active-directory-security-trimming", "type" : "active-
directory-security-trimming" }
```

Configuration

	When entering configuration values in the UI, use *unescaped* characters, such as `\t` for the tab character. When entering configuration values in the API, use *escaped* characters, such as `\\t` for the tab character.

Additional Query Parameters Stage

The Additional Query Parameters query pipeline stage is used to set, append, and remove Solr query parameters *(https://wiki.apache.org/solr/QueryParametersIndex)*. This stage takes a list of query parameters. Each parameter is specified as a triple consisting of parameter name, parameter value, and update policy.

Available update policies are: `replace`, `append`, `remove` and `default`. The policy 'default' means that this parameter will be added only if it has not yet been set via the request or by a default specification in the Solr config file solrconfig.xml.

Example Stage Specification

Set the response format to JSON:

```
{ "type":"set-params", "id":"default-params", "params": [ {"key":"wt",
"value":"json", "policy":"default"} ], "skip":false }
```

Return a field named "id", and 10 rows of results:

```
{ "type":"set-params", "id":"default-params", "params": [ {"key":"fl",
"value":"id", "policy":"append"}, {"key":"rows", "value":"10", "policy":"replace"}
], "skip":false }
```

Configuration

When entering configuration values in the UI, use *unescaped* characters, such as `\t` for the tab character. When entering configuration values in the API, use *escaped* characters, such as `\\t` for the tab character.

Analytics Catalog Stage

The Analytics Catalog query pipeline stage lets you define views of data stored in Fusion collections. You can query the objects in the Analytics Catalog using SQL, Solr Streaming Expressions *(https://cwiki.apache.org/confluence/display/solr/Streaming+Expressions)*, or regular Solr queries.

Configuration

When entering configuration values in the UI, use *unescaped* characters, such as `\t` for the tab character. When entering configuration values in the API, use *escaped* characters, such as `\\t` for the tab character.

Apply Rules Stage

This stage looks up rules that have been deployed to the *`COLLECTION_NAME_query_rewrite` collection* and matches them against the query. Matching rules that perform *query rewriting* are applied at this stage, while matching rules that perform *response rewriting* are applied by the *Modify Response with Rules* stage later in the pipeline.

To trigger a rule that contains a tag, specify the tagname in the request URL of the user search app. See *Easily define triggers in tags* for more information.

See *Rules* for additional details about how to configure this set of features.

This stage supports *asynchronous processing*.

Although this stage is available without a Fusion license, it is only effective after creating rules in Fusion.

Configuration

When entering configuration values in the UI, use *unescaped* characters, such as `\t` for the tab character. When entering configuration values in the API, use *escaped* characters, such as `\\t` for the tab character.

Block Documents Stage

The Block Documents query pipeline stage removes documents from the result based on a Block Documents rule which consists of the following elements:

- `field`. The document field to filter on.
- `keywords`. The words, phrases, or regex used as the filter.
- `mode`. Filtering logic applied to keywords, one of:
  - `exact`. exact matching on any item in the keywords list.
  - `phrase`. phrase matching on the items in the keywords list.
  - `regex`. treat items in the keywords list as a regex.
  - `match`. requires a match for every item in the keyword list, but does not require phrase matching.
- `blocks`. A list of document IDs for documents which are always removed from the query result.

The block documents rule is used to craft a Solr query. The `keywords` are added to the `q` Solr query parameter, by default. The configuration property `queryParam` can be used to specify a different Solr query parameter to use as the keywords filter. The rest of the rule is processed into the `fq` Solr query parameter.

Configuration

	When entering configuration values in the UI, use *unescaped* characters, such as `\t` for the tab character. When entering configuration values in the API, use *escaped* characters, such as `\\t` for the tab character.

Boost Documents Stage

The Boost Documents query pipeline stage adds boosting parameters to matched documents based on user-defined rules. Boosts are defined with a term value to boost and the boost factor to add. The boosting parameters are added to the `bq` Solr query parameter.

The Boost Documents rule consists of the following elements:

- `field`. The document field to filter on.
- `keywords`. The words, phrases, or regex used as the filter.
- `mode`. Filtering logic applied to keywords, one of:
  - `exact`. The keyword and the query must match exactly. This is case-sensitive.
  - `phrase`. The phrase matching on the items in the keywords list.
  - `regex`. Treat items in the keywords list as a regex.
  - `match`. Must match every item in the keyword list, but does not require phrase matching.
- `boosts`. Consists of a list of pairs.
  - `value`. One or more terms used for boosting.
  - `boost`. The numeric boost value. Default `100`.

Configuration

	When entering configuration values in the UI, use *unescaped* characters, such as `\t` for the tab character. When entering configuration values in the API, use *escaped* characters, such as `\\t` for the tab character.

Boost calculation

In Fusion 5.x.x, numbers entered into boosts are *multiplied* with the score.

	This differs from Fusion 4.x.x where numbers entered into boosts are *added* to the score.

As a result of multiplication:

- Negative boosts are not possible. You must use fractional values to deprioritize the score. For example, a boost value of 0.5 decreases the score by half.
- Do not use values such as 100, which was considered reasonable in Fusion 4.x.x. If those values are entered in Fusion 5.x.x, document scores are boosted to an unrealistically high level.
- Fusion generates an error when a query is executed on a pipeline that uses negative boost values.

Boost with Signals Stage

The Boost with Signals query pipeline stage uses *aggregated signals* to selectively boost items in the set of search results.

Using the main query and the stage configuration parameters, this stage performs a secondary query to the `COLLECTION_NAME_signals_aggr` collection and returns updated boost weights for the items in the main query's search results. Items that have received more user interaction also receive higher boost weights.

See *Recommendation Methods* for more information.

This stage supports *asynchronous processing*.

> This stage accesses the `signals_aggr` collection. Before using it, verify that the following *permission* is set: `GET:/solr/COLLECTION_NAME_signals_aggr/select`

Configuration overview

The fields below are especially useful to understand when configuring this stage.

Number of Recommendations `numRecommendations`	Sets the `rows` query param in the main query as the maximum number of query results which will be boosted by this pipeline stage.
Number of Signals `numSignals`	Sets the `rows` query param in the query that searches the `COLLECTION_NAME_signals_aggr` collection, so only the specified number of aggregated signals are retrieved and used for boosting. When signals boosting is applied to a query, aggregated signals records are queried from the appropriate `_signals_aggr` collection to find out the popularity or boost weight for documents which have signals. `numSignals` limits the number of records to be queried from a `}}{{_signals_aggr` collection and used to calculate this boost.
Aggregation Type `aggrType`	A filter to retrieve aggregated signals in the `COLLECTION_NAME_signals_aggr` collection per each aggregated signal's `aggr_type_s` field value.
Solr Field to Boost On `boostId`	The document field in the main collection on which to perform boosting. Typically it should use default field, which is `id`. This field corresponds to the **Rollup Field**/`rollupField` field. Together, these two fields act like a `FIELD:VALUE` pair in the query modification for boosting.

Boost Method
 	
`boostingMethod`
	This adds a query parameter to the original query, either "query-param" or "query-parser". The result is `("query-param" or "query-parser") + Boost Param("boost" or `bq`)`, as in the examples below:
	
	"query-param"+"boost", result boost query param
	
	`boost="map(query({!field f='id' v='6239046,13026192}), 0, 0, 1, 27.1705)"`
	
	"query-parser"+"boost", result boost query param
	
	`bp_xxx_bbqx="map(query({!field f='id' v='6239046,13026192}), 0, 0, 1, 27.1705)"`
	
	When `Boost Param` uses `bq`, similar logic applies. When **Boost Param**/`boostingParam` uses "boost", it works with both "query-param" and "query-parser".

Rollup Field `rollupField`	Indicates which aggregated signal document field the boost parameter will use for the final boosting. It works in combination with the **Solr Field to Boost On**/`boostId` field. This should be set to the field in the aggregated signal collection that stores the doc list that is aggregated as one record. By default it's set to `doc_id_s`, because by default this is used in the *click_signal_aggr SQL job*.

Rollup Weight Field `rollupWeightField`	Indicates the final boost weight used to calculate the new score for docs retrieved by the main query. Similar to **Rollup Field**/`rollupField` above, this should be set to the field in the aggregated signal collection that stores the final weight that was calculated. By default it's `weight_d`, because by default this is used in the *click_signal_aggr SQL job*.
Final Boost Weight Expression `weightExpression`	Calculates the final weight using the weight and score retrieved from the `COLLECTION_NAME_signals_aggr` collection. The default value is `math:log(weight_d + 1) + 10 * math:log(score+1)`.

Solr query parameters

These parameters are used in the **Solr Query parameters**/`queryParams` field for retrieving signal aggregation docs from the `COLLECTION_NAME_signals_aggr` collection. These Solr query params will affect which aggregated signals are used for producing the boosting parameter on the main query.

`qf=query_t`	Defines which field to query. In the default case, the query searches on the `query_t` field of aggregated signal docs.
`pf=query_t^50`	Boosts docs within the set of retrieved docs using phrase matching.
`pf2=query_t^20`	`pf2` is similar to `pf`; the difference is that `pf2` works on bigram phrases.
`pf3=query_t^10`	`pf3` is similar to `pf`; the difference is that `pf3` works on trigram phrases.

FAQs

If there is `fq` in the main query, how is it matched with the correct aggregated signal?
In this case, you need to use the `lw.rec.fq` query parameter in the main query. `lw.rec.fq` can be parsed by the Boost with Signals stage, and therefore the filters specified in it can be added to the Solr query that is retrieving the aggregated signals.

For example, if we have filter query param `fq=format:CD&fq=name:Latin`, this needs to be translated into `lw.rec.fq=filters_s:"format:cd $ name:latin"`. Values must be lowercase. The final main query should be:

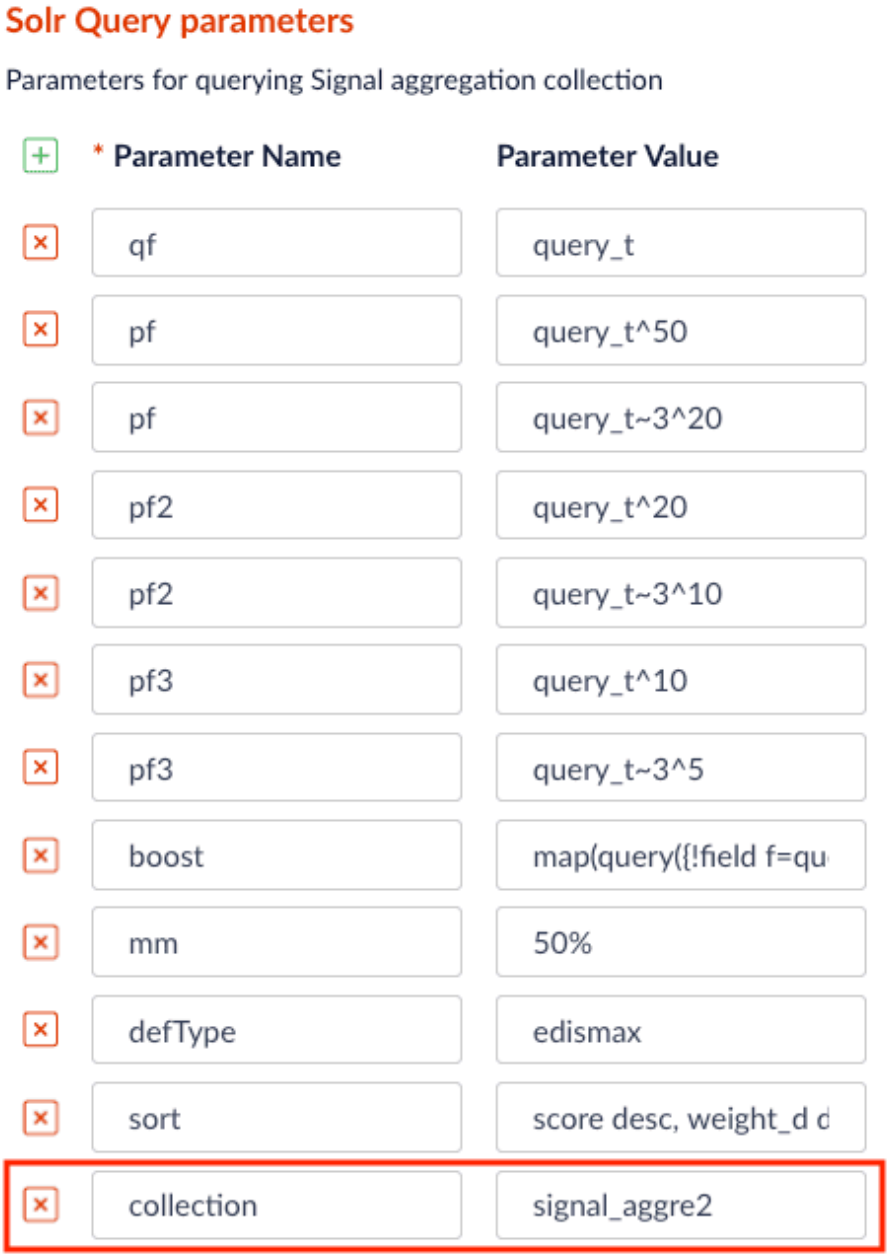
```
http://FUSION_HOST:FUSION_PORT/api/apps/demo_app/query-pipelines/demo_app/collections/demo_app/select?echoParams=all&wt=json&json.nl=arrarr&sort&start=0&q=apple&debug=true&rows=10&lw.rec.fq=filters_s:"format:cd $ name:latin"
```

Now the Boost with Signals stage will only retrieve aggregated signals that have the same filter query.

> If there are multiple `fq` values (for example, `format:cd` and `name:latin`), they are ordered alphabetically as strings and joined with " $ " (a $ with a space on each side). In the example, `"format:cd $ name:latin"`.

What if my aggregated signals are in a different collection?

You can point the Boost with Signals stage to a different signal collection by adding a `collection` parameter in the `Solr Query Parameters` section.



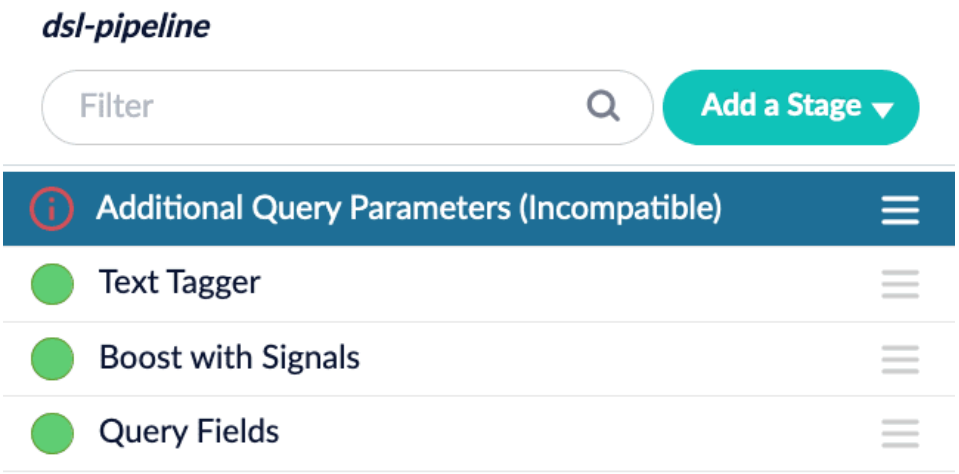
Configuration

	When entering configuration values in the UI, use *unescaped* characters, such as `\t` for the tab character. When entering configuration values in the API, use *escaped* characters, such as `\\t` for the tab character.

DSL to Legacy Parameters Stage

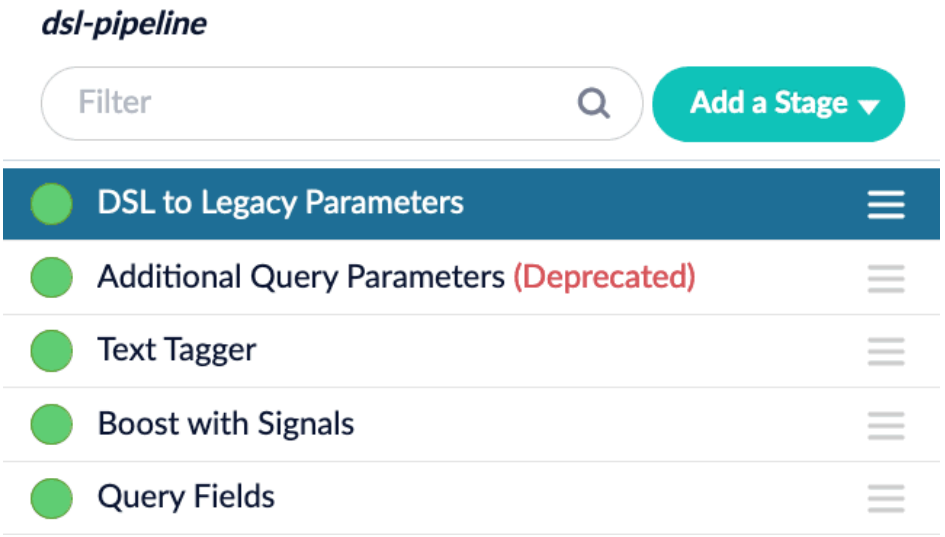
Fusion Domain Specific Language (DSL) provides expected search results as a JSON response in a way that reduces search query complexity for the user. Not all query pipeline stages support DSL. The **DSL to Legacy Parameters** query pipeline stage allows you to convert your DSL requests to the legacy (Solr) format.

If you add an incompatible stage while in DSL mode, Fusion flags the stage:



The **DSL to Legacy Parameters** stage extends DSL support to incompatible stages. Click **Add a Stage** and select the **DSL to Legacy Parameters** stage. This stage converts the DSL request to legacy parameters for *subsequent* stages, so it can be processed by the incompatible stage.

	The **DSL to Legacy Parameters** stage must come before incompatible stages for the conversion to succeed.



See the *DSL documentation* for more details on what you can do with DSL.

Configuration

	When entering configuration values in the UI, use *unescaped* characters, such as `\t` for the tab character. When entering configuration values in the API, use *escaped* characters, such as `\\t` for the tab character.

Experiment Stage

The Experiment query pipeline stage is part of Fusion's *Machine Learning* framework.

The Machine Learning framework provides tools to evaluate alternative methods of computing and displaying search results. For example, an *experiment* may consist of a system which has two or more different query pipelines running, and users are randomly served search results using some variant of the system via instrumented pages that capture user response. In such a system, the Experiment query stage selects a variant from a running experiment and inject its properties into the current *Context*.

Configuration

> When entering configuration values in the UI, use *unescaped* characters, such as `\t` for the tab character. When entering configuration values in the API, use *escaped* characters, such as `\\t` for the tab character.

Field Facet Stage

The Field Facet query pipeline stage is used to add a Solr Field Facet query *(https://cwiki.apache.org/confluence/display/solr/Faceting#Faceting-Field-ValueFacetingParameters)* to the search query pipeline.

A field facet query computes the top values for a field and returns the list of those values along with a count of the subset of documents in the search results which match that term. Field faceting works best over fields which contain a single label or set of labels from a finite, controlled lexicon such as product category. Facet field parameters can be tuned for performance, see: Facet Field Configuration *(https://cwiki.apache.org/confluence/display/solr/Faceting#Faceting-Field-ValueFacetingParameters)*.

It is possible to specify more than one field facets. For each field facet you must specify the field name plus the following additional parameters:

- **Limit.** The maximum number of terms to be returned. Default 100.
- **Offset.** The number of top facet values to skip in the response. Default 0.
- **Sort.** The order in which to list facet values: `count` ordering is by documents per term, descending, and `index` ordering is sorted on term values themselves.
- **Missing.** The number of documents in the results set which have no value for the facet field.
- **Choice of facet method (advanced).** Specify Solr algorithm used to calculate facet counts. (See Facet Method Configuration *(https://cwiki.apache.org/confluence/display/solr/Faceting#Faceting-Thefacet.methodParameter)* for details). One of:
  - `enum`. Small number of distinct categories.
  - `fc` ("field cache"). Many different values in the field, each document has low number of values, multi-valued field.
  - `fcs` ("single value string fields"). Good for rapidly changing indexes.

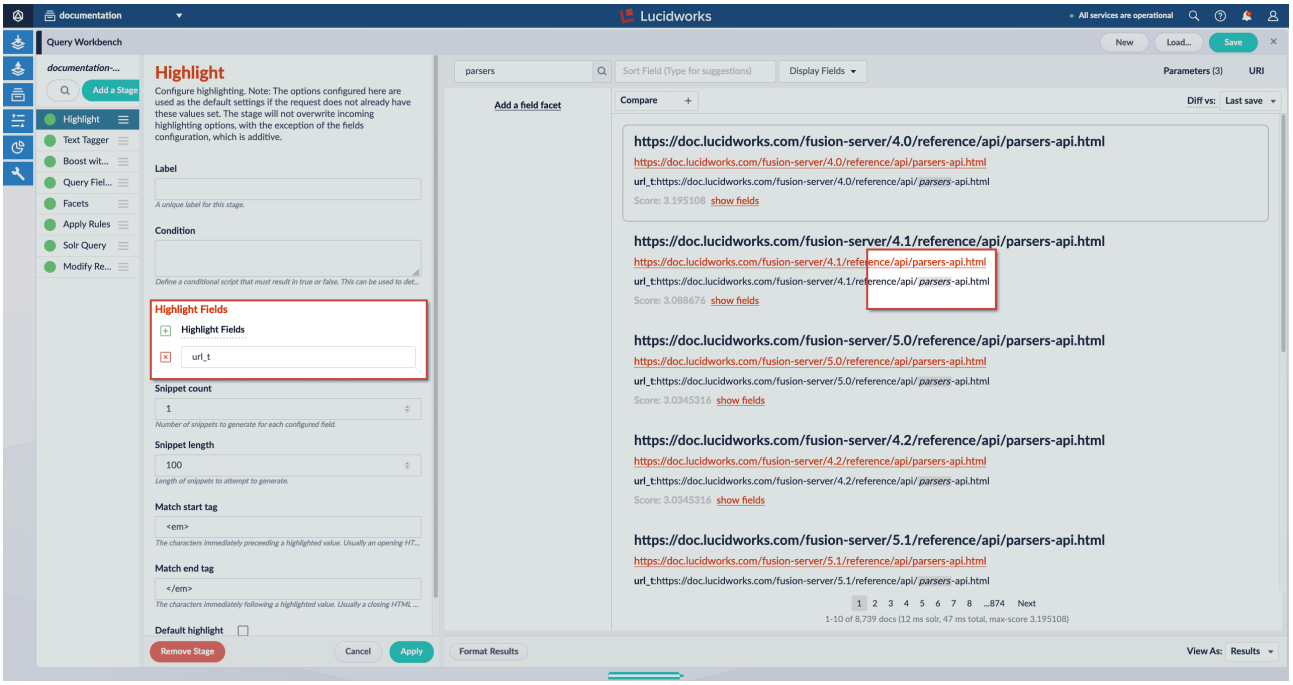
For further details see: Solr Faceting Overview *(https://lucene.apache.org/solr/guide/faceting.html)*.

Configuration

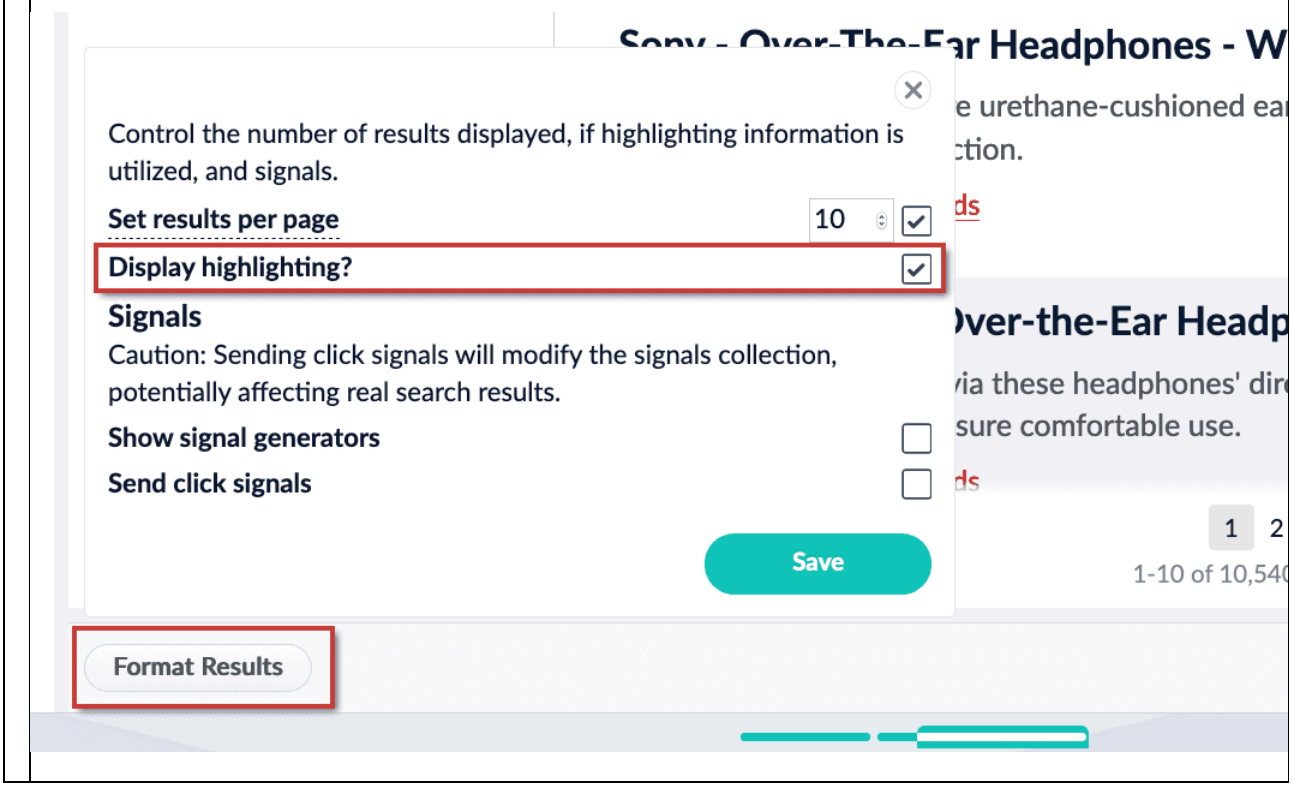
Highlight Stage

Overview

The Highlight stage allows you to highlight key parts of fields by generating snippets and appending match tags to the highlighted terms. The highlight match tags can then be styled in your web search application.



To see highlighting in the Query Workbench, turn it on by setting **Format Results** > **Display highlighting?** to `true`:



How does this differ from other highlight methods?

- **Highlight Query Pipeline Stage.** Introduced in Fusion 5.2.0, the *Query Highlight pipeline stage* allows you to configure standard options for highlighting using Solr's Unified highlighter *(https://solr.apache.org/guide/8_6/highlighting.html#the-unified-highlighter)* instead of adding Solr request parameters. For instructions, see *Configure the Highlight Query Pipeline Stage*.
- **Additional Query Parameters stage.** For Fusion 5.2.0 and later, the same method used in Fusion 5.1.x can be used. Configure highlighting with the *Additional Query Parameters stage*. For instructions, see *Configure Highlighting in the Query Workbench*.

Parameters

Highlight Fields

Specifies the fields to use for highlighting. Highlighting should be used *for text fields only*. These fields are typically identified by the `_t` suffix, for example: `longDescription_t`.

Snippet count

Specifies the number of snippets to generate for each configured field. When **Snippet count** is assigned a value of `3`, for example, the following is returned:

```
"highlighting": { "link:/fusion/5.2/concepts/signals-and-aggregations/signals/": {
"body_t": [ "Click <em>signals</em> are the most common type of <em>signals</em> as
this is the most common action a user takes with an item. ", // 1 "The
<em>signals</em> collection is created automatically when <em>signals</em> are
enabled for the primary collection. ", // 2 "App Insights mainly uses raw
<em>signals</em>, but also uses some aggregated <em>signals</em>. " // 3 ] } }
```

Snippet length

Controls the length of snippets. The **Snippet length** value is a character count. For example, setting **Snippet count** to `3` and **Snippet length** to `5` will attempt to generate 3 snippets that are at least 5 characters long:

```
"highlighting": { "link:/fusion/4.1/concepts/signals-and-aggregations/signals/": {
"body_t": ["Click <em>signals</em> are the most common type of <em>signals</em> as
this is the most common action a user takes with an item. ", // 1 "The
<em>signals</em> collection is created automatically when <em>signals</em> are
enabled for the primary collection. ", // 2 "App Insights mainly uses raw
<em>signals</em>, but also uses some aggregated <em>signals</em>. " // 3 ] } }
```

In the example above, 3 snippets were generated, each snippet was at least 5 characters long, and snippets returned as a complete sentence. Setting the **Snippet length** to a higher value, such as `500`, may result in a single long snippet despite a **Snippet count** value of `3`:

```
"highlighting": { "link:/fusion/5.2/concepts/signals-and-aggregations/signals/": {
"body_t": ["<em>Signals</em> are indexed in a secondary collection which is linked
to the primary collection by the naming convention <primarycollectionname>_signals.
So, if your main collection is named products, the associated <em>signals</em>
collection is named products_signals. The <em>signals</em> collection is created
automatically when <em>signals</em> are enabled for the primary collection.
<em>Signals</em> are enabled by default whenever a new collection is created.
<em>Signals</em> are indexed just like ordinary documents. The <em>signals</em>
collection can be searched like any other collection, for example by using the
Query Workbench with the <em>signals</em> collection selected. App Insights
provides visualizations and reports with which to analyze your <em>signals</em>.
App Insights mainly uses raw <em>signals</em>, but also uses some aggregated
<em>signals</em>. Currently only the signal types Request, Response and Click are
supported within the App Insights dashboards. See the descriptions of
<em>signals</em> types and structure for more information. "] } }
```

Match tags

Specifies the tags to use for identifying highlighted portions of the snippet. By default `<em>` and `</em>` are used. Match tags are appended to the highlight response field:

JSON

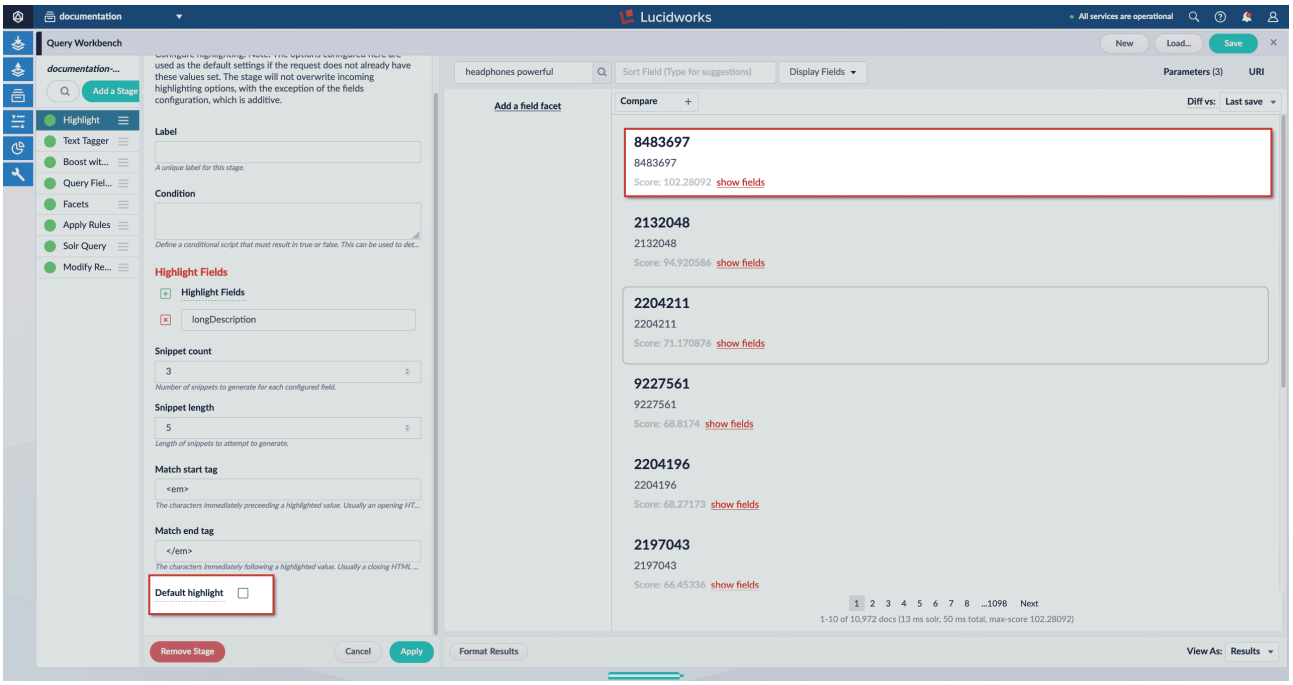
```
"highlighting" : { "8483697" : { "longDescription_t" : [ "Use these <em>powerful</em> headphones to stay focused on the music and help tune out background noise." ] }, "2132048" : { "longDescription_t" : [ "Enjoy the beats and rhythms of your favorite music with these stylish over-the-ear headphones that feature 40mm drivers for clear and <em>powerful</em> sound. " ] } }
```

You can use other values for the match tags, such as `<span style="background-color:powderblue;">` and `</span>`, but the Fusion UI will not reflect your custom styles.

Default highlight

If no highlighting match is found, the results are returned without highlighting. This is easiest to see in the JSON. In the examples below, `longDescription_t` is as the **Highlight Fields** value.

UI

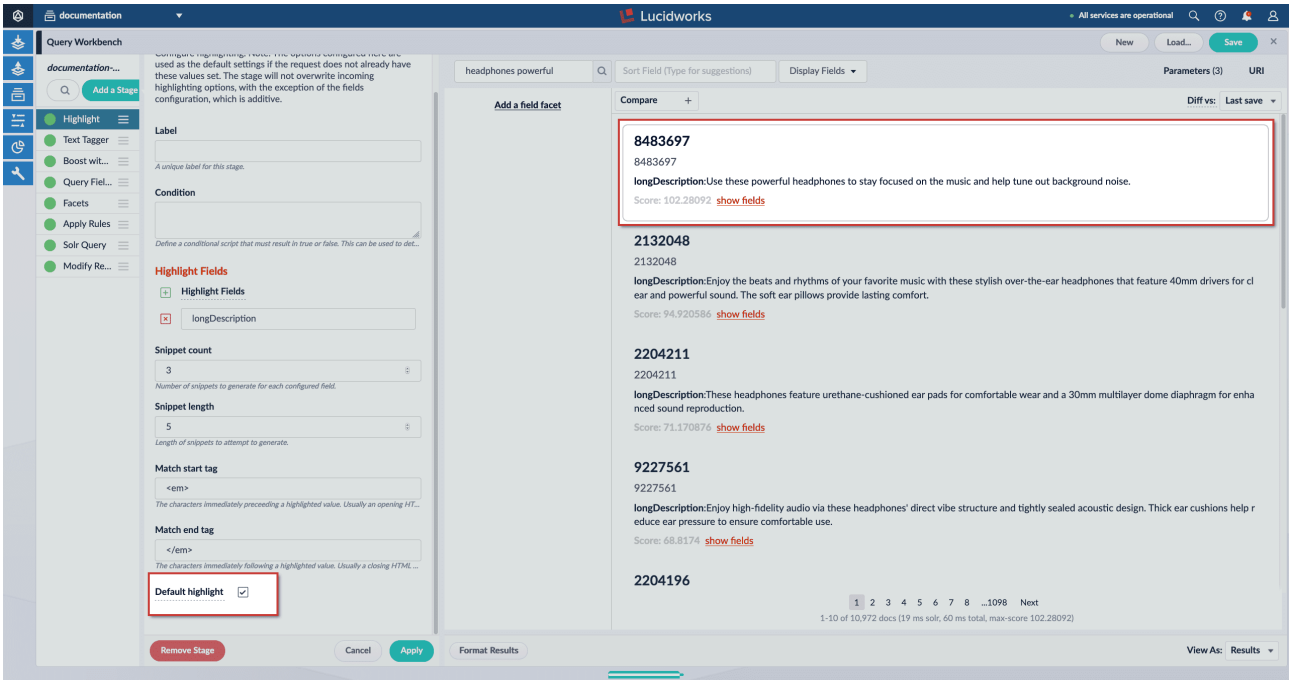


JSON

```
"highlighting" : { "8483697" : { "longDescription_t" : [ ] }, "2132048" : { "longDescription_t" : [ ] } }
```

In the examples above, no highlight match is found for the documents. Turn on **Default highlight** to force the stage to return the leading part of the field instead. Use this option to avoid returning large fields if you need to display something even when there is no highlight match.

UI



JSON

```
"highlighting" : { "8483697" : { "longDescription_t" : [ "Use these powerful
headphones to stay focused on the music and help tune out background noise." ] },
"2132048" : { "longDescription_t" : [ "Enjoy the beats and rhythms of your favorite
music with these stylish over-the-ear headphones that feature 40mm drivers for
clear and powerful sound. " ] } }
```

Configuration

	When entering configuration values in the UI, use *unescaped* characters, such as `\t` for the tab character. When entering configuration values in the API, use *escaped* characters, such as `\\t` for the tab character.

JDBC Lookup Stage

The JDBC Lookup query pipeline stage is used to call out to a database as part of a pipeline stage, to inject results into either the context/request or the pipeline document. For example, if you needed to look up a user from a DB and add their profile information onto a request for downstream use in a pipeline, the JDBC Lookup Stage would facilitate this.

	You must first *upload the JDBC driver to Fusion*.

This stage supports *asynchronous processing*.

Example

An example of setup for a JDBC Lookup query pipeline stage.

Upload stage config via POST to Fusion REST API endpoint `api/query-stages/instances`

```
curl -u USERNAME:PASSWORD -X POST -H "Content-Type: application/json" -d '{"id":
"jdbc-quer","type":"jdbc-query-lookup","driver":"postgresql-9.3-1101-
jdbc4.jar","connection":"jdbc:postgresql:database","username":"user","password":"pa
ssword1","preparedStatement":"select ID as id from DATABASE;"}'
http://FUSION_HOST:FUSION_PORT/api/query-stages/instances
```

Response

```
{ "type" : "jdbc-query-lookup", "id" : "jdbc-quer", "driver" : "postgresql-9.3-
1101-jdbc4.jar", "connection" : "jdbc:postgresql:database", "username" : "user",
"password" : "password1", "preparedStatement" : "select ID as id from DATABASE;",
"fetchSize" : -1, "join" : true, "rows" : 10, "skip" : false, "label" : "jdbc-
query-lookup", "type" : "jdbc-query-lookup" }
```

Configuration

	When entering configuration values in the UI, use *unescaped* characters, such as `\t` for the tab character. When entering configuration values in the API, use *escaped* characters, such as `\\t` for the tab character.

JavaScript Query Stage

The JavaScript Query stage allows you to write custom processing logic using JavaScript to manipulate search requests and responses. The first time that the pipeline is run, Fusion compiles the JavaScript program into Java bytecode using the JDK's JavaScript engine.

The JavaScript Query stage allows you to run JavaScript functions over search requests and responses by manipulating variables called "request" and "response" which are *Request objects* and *Response objects*, respectively.

	Users who can create or modify code obtain access to the broader Fusion environment. This access can be used to create intentional or unintentional damage to Fusion.

The JavaScript Engine Used by Fusion

The JavaScript engine used by Fusion is the Nashorn engine from Oracle. See The Nashorn Java API *(https://docs.oracle.com/javase/8/docs/technotes/guides/scripting/nashorn/api.html)* for details.

JavaScript Query Stage Global Variables

JavaScript *(http://en.wikipedia.org/wiki/JavaScript)* is a lightweight scripting language. The JavaScript in a JavaScript stage is standard ECMAScript *(http://en.wikipedia.org/wiki/ECMAScript)*. What a JavaScript program can do depends on the container in which it runs. For a JavaScript Query stage, the container is a Fusion query pipeline. The following global pipeline variables are available:

Name	Type	Description
`request`	*Request for a Regular Query*	The `request` variable contains Solr query information and is referred to as a **regular** request. A regular query request does *not* include Search DSL (domain specific language) parameters.
`request`	*DSL Request*	The `request` variable is also used when the query contains parameters for a Search DSL (domain specific language) request. See *Domain Specific Language* for more information.
`response`	*Response for a Regular Query*	The `response` variable contains Solr response information, and is used to return information from a **regular** request. Because a regular request does *not* include Search DSL (domain specific language) parameters, the response will not return Search DSL results.
`response`	*DSL Response*	The `response` variable is also used to return information from a Search DSL (domain specific language) request. See *Domain Specific Language* for more information.
`ctx`	*Context*	A map that stores miscellaneous data created by each stage of the pipeline. Use the `ctx` variable instead of the deprecated `_context` global variable. The `ctx` variable is used to: • Pass data from one stage to another • Store data that needs to be passed from one custom stage to a later custom stage The data can differ between stages: • If the previous stage changes the data • Based on the configuration of each stage If the data is modified in one stage, it may cause a later stage to function irregularly.
`collection`	String	The name of the Fusion collection being indexed or queried.
`solrServer`	*BufferingsolrServer*	The Solr server instance that manages the pipeline's default Fusion collection. All indexing and query requests are done by calls to methods on this object. See solrClient *(https://lucene.apache.org/solr/5_2_1/solr-solrj/org/apache/solr/client/solrj/SolrClient.html)* for details.
`solrServerFactory`	*solrClusterComponent*	The SolrCluster server used for lookups by collection name which returns a Solr server instance for that collection, e.g. `var productsSolr = solrServerFactory.getSolrServer("products");`.
`QueryRequestAndResponse`	*QueryRequestAndResponse*	Used to create query pipeline requests and responses.
`JSONResponse`	*JSONResponse*	Returns list of info as a string.

See *Custom JavaScript Query Stage Examples* for more information.

Syntax Variants

JavaScript stages can be written using legacy syntax or function syntax. The key difference between these syntax variants is how the "global variables" are used. While using legacy syntax, these variables are used as global variables. With function syntax, however, these variables are passed as function parameters.

Legacy Syntax

Legacy syntax is used to perform very simple tasks.

```
//do some work... return doc;
```

Function Syntax

```
function(request,response) { request.addParam("foo", "bar"); }
```

	Function syntax is used for the examples in this document.

Global variable `logger`

The logs are output to the query service logs for custom query stages. Access the Log Viewer and filter on this service to view the information.

Configuration

	When entering configuration values in the UI, use *unescaped* characters, such as `\t` for the tab character. When entering configuration values in the API, use *escaped* characters, such as `\\t` for the tab character.

Landing Pages Stage

The Landing Pages query pipeline stage provides the mechanism by which specific search queries will be pinned to certain URLs. This stage returns one or more URLs which can be used for redirection. It does not preform a redirection; this must be done by the calling application. The redirection URLs are returned in a separate section of the Fusion response object, with attribute name `fusion`.

This stage is configured using Landing Page Rules, which consist of the following:

- `keyword`. Words, phrases, or a regex.
- `mode`. Filtering logic applied to the query, one of:
  - `exact`. The keyword and the query must match exactly. This is case-sensitive.
  - `phrase`. The phrase matching on the items in the keywords list.
  - `regex`. Treat items in the keywords list as a regex
  - `match`. Must match every item in the keyword list, but does not require phrase matching.
- `url`. A list of URLs.

Configuration

	When entering configuration values in the UI, use *unescaped* characters, such as `\t` for the tab character. When entering configuration values in the API, use *escaped* characters, such as `\\t` for the tab character.

Logging Stage

The Logging query pipeline stage prints messages to the API log file; the default location is: `https://FUSION_HOST:FUSION_PORT/var/log/api/api.log`.

The verbosity of this message is controlled by the property `detailed`. If true, then the current Request object will be pretty-printed to the log file. If false, only the basic information about this stage will be logged.

In a production environment logging stages should be configured with property `skip` set to `true`, if possible. Use of detailed logging may impact performance.

Configuration

	When entering configuration values in the UI, use *unescaped* characters, such as `\t` for the tab character. When entering configuration values in the API, use *escaped* characters, such as `\\t` for the tab character.

Machine Learning Stage

The Machine Learning query pipeline stage uses a trained machine learning model to analyze a field or fields of a *Request* object and stores the results of analysis in a new field added to either the Request or the *Context* object.

In order to use the Machine Learning Stage, you must train a machine learning model. There are two different ways to train a model:

- Use a Fusion job that trains a model, like *Classification*
- *Develop and Deploy a Machine Learning Model*

This stage requires that you use JavaScript to construct a model input object from the Request and/or Context. This JavaScript is defined in the "Model input transformation script" property. This script must construct a HashMap containing fields and values to be sent to the model. The field names and values will depend on the input schema of the model.

Value types supported are:

- `String`
- `Double`
- `String[]`
- `double[]`
- `List<String>`
- `List<Number>`

The JavaScript interpreter that executes the script will have the following variables available in scope:

- *`request`*
- *`response`*
- *`context`*

The last line of the script must be a reference to the HashMap object you created.

Example 1: Single string parameter from request to modelInput HashMap

```
var modelInput = new java.util.HashMap() modelInput.put("input_1",
request.getFirstParam("q")) modelInput
```

Example 2: List of strings from request to modelInput HashMap

```
var modelInput = new java.util.HashMap() modelInput.put("input_1",
request.getParam("q")) // request.getParam returns a Collection modelInput
```

Example 3: List of numeric values from request to modelInput HashMap

```
var modelInput = new java.util.HashMap() var list = new java.util.ArrayList()
list.add(Double.parseDouble(request.getFirstParam("numeric_1")))
list.add(Double.parseDouble(request.getFirstParam("numeric_2")))
modelInput.put("input_1", list) modelInput
```

Similarly, you will need to use JavaScript to store the predictions into the Request and/or Context from the model output object. The model output object is a HashMap containing fields and values produced by the model.

The JavaScript interpreter that executes the script will have the following variables available in scope:

- modelOutput (a HashMap containing fields/values returned from ML model service)
- *`request`*

- `response`
- `context`

Example: Place predictedLabel (string) on request

```
request.putSingleParam("sentiment", modelOutput.get("predictedLabel"))
```

Configuration

Managed JavaScript Query Stage

This stage manipulates a query using a script from the *blob store*. The script must have the blob type `file:js-query`.

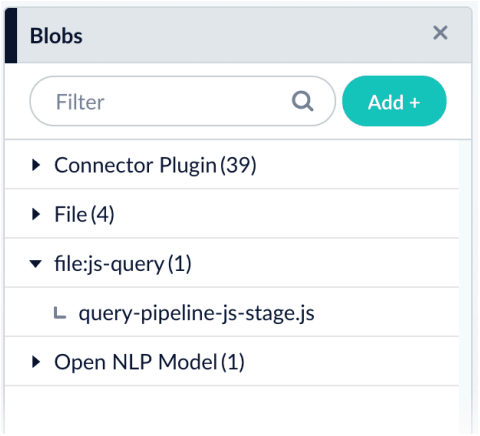
See also: *Custom JavaScript Stages For Query Pipelines*.

Uploading a JavaScript blob

In Fusion 5.4, JavaScript blobs must be uploaded using the *Blob Store API*, like this:

```
curl -u USERNAME:PASSWORD -X PUT -H 'Content-Type: text/javascript' --data-binary
@query-pipeline-js-stage.js
http://FUSION_HOST:FUSION_PORT/api/apps/APP_NAME/blobs/query-pipeline-js-stage.js?
resourceType=file:js-query
```

Open the *blob manager* to verify that the new blob is present:



Referencing a JavaScript blob

Use any of the following formats for the `ref`/**Script Reference** configuration field:

```
query-pipeline-js-stage.js
```

```
blob:query-pipeline-js-stage.js
```

```
urn:x-fusion:blob:query-pipeline-js-stage.js
```

The examples above use `query-pipeline-js-stage.js` as the blob ID.

Edit JavaScript

For Fusion 5.4.x and later, select **Edit** in the **Script Reference** field to edit JavaScript directly in the Fusion UI.

Configuration

When entering configuration values in the UI, use *unescaped* characters, such as `\t` for the tab character. When entering configuration values in the API, use *escaped* characters, such as `\\t` for the tab character.

Merge Async Results Stage

Wait for previous *asynchronous stages* to complete and merge their results into the active request.

Configuration

When entering configuration values in the UI, use *unescaped* characters, such as `\t` for the tab character. When entering configuration values in the API, use *escaped* characters, such as `\\t` for the tab character.

Milvus Ensemble Query Stage

The Milvus Ensemble Query stage performs ensemble of different search results to retrieve metadata from Solr (see *Milvus documentation*).

Configuration

When entering configuration values in the UI, use *unescaped* characters, such as `\t` for the tab character. When entering configuration values in the API, use *escaped* characters, such as `\\t` for the tab character.

Milvus Query Stage

The Milvus Query stage performs vectors similarity search in *Milvus*.

Configuration

When entering configuration values in the UI, use *unescaped* characters, such as `\t` for the tab character. When entering configuration values in the API, use *escaped* characters, such as `\\t` for the tab character.

Milvus Response Update Stage

The Milvus Response Update stage updates response documents with vectors similarity and ensemble scores (see *Milvus documentation*). It should be used after the *Milvus Ensemble Query stage* and after the *Solr Query stage*.

Configuration

When entering configuration values in the UI, use *unescaped* characters, such as `\t` for the tab character. When entering configuration values in the API, use *escaped* characters, such as `\\t` for the tab character.

Modify Response with Rules Stage

Most rules operate on the request, but some *rule types*, such as banner rules or redirect rules, do their work when the response comes back. The Modify Response with Rules stage applies those rules to the response. For example, a banner rule can add a banner URL to the response before returning it to the client.

See *Rules* for additional details about how to configure this set of features.

TIPS:

- The *Apply Rules* stage must be enabled as a prerequisite for this stage.
- This stage has no configuration keys; simply enable or disable the stage.
- Although this stage is available without a Fusion license, it is only effective after creating *rules* in Fusion.

Configuration

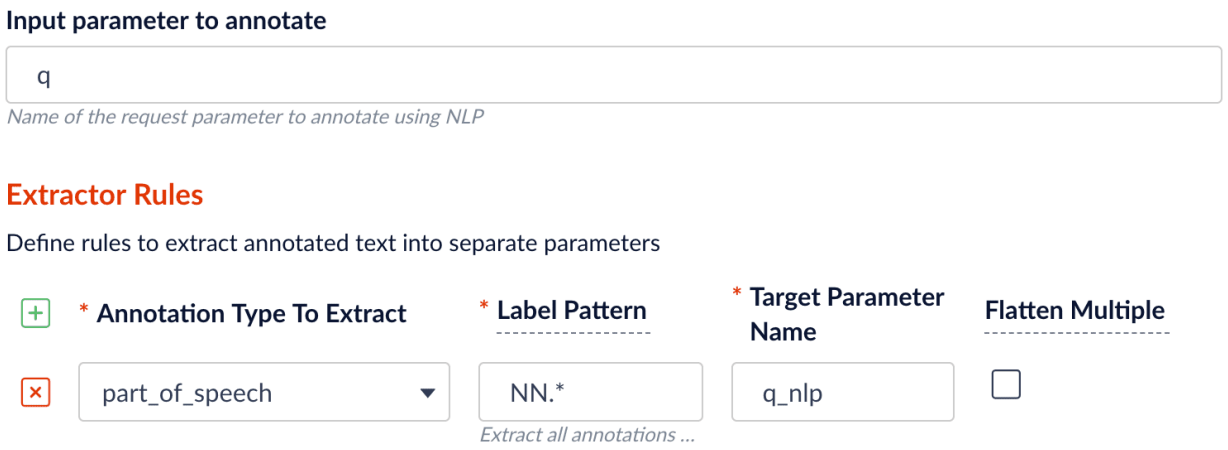
When entering configuration values in the UI, use *unescaped* characters, such as `\t` for the tab character. When entering configuration values in the API, use *escaped* characters, such as `\\t` for the tab character.

NLP Annotator Query Stage

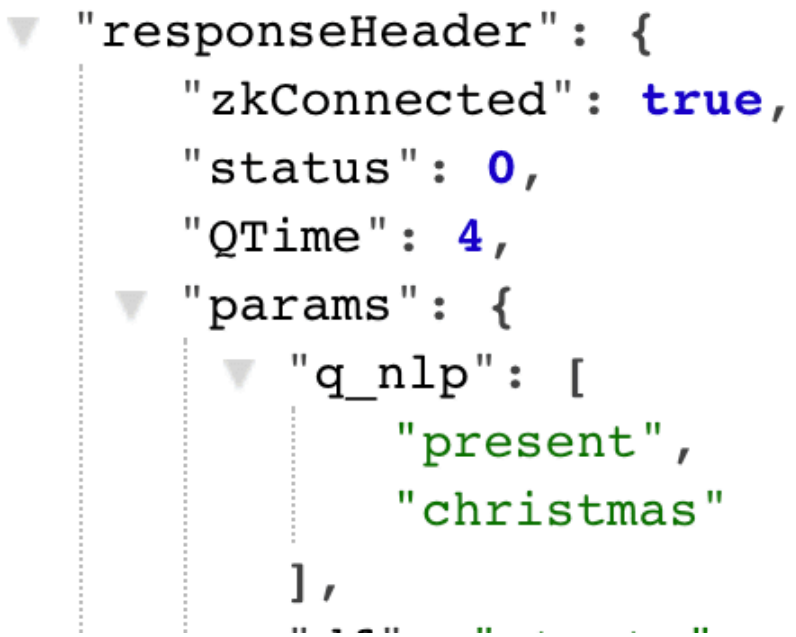
Like the *NLP Annotator index stage*, the NLP Annotator query stage can be included in an query pipeline to perform *Natural Language Processing* tasks.

Example of how to use NLP Annotator Query stage:

1. Add the NLP Annotator query stage to the query pipeline.
2. Supply the Model ID ("opennlp", "spacy", or the model ID given to the uploaded Spark NLP model).
3. Specify the input parameter, label pattern and target parameter fields:



- input parameter field: the Fusion query parameter text, normally `q` since we want to annotate the raw query string to understand the intent.
- label pattern: regex pattern that matches the NER/POS labels: for example, `PER.` **will match extracted name entities with label** `PERSON`**, while** `NN.` will match tagged nouns.
- target parameter field: the outcome extraction/tagging, and.
  For the query stage, the result is set to be put in a new query parameter field:



Configuration

	When entering configuration values in the UI, use *unescaped* characters, such as `\t` for the tab character. When entering configuration values in the API, use *escaped* characters, such as `\\t` for the tab character.

Parameterized Boosting Stage

The Parameterized Boosting query pipeline stage reads the `boostValues` (in `List<DocumentResult>` format) from the context variable (added by a *Rollup Aggregation stage* or a *JavaScript stage*), and adds boosts to the main query using `bq` or `boost` based on the stage configuration. The weights for the boost values can also be scaled.

Configuration

When entering configuration values in the UI, use *unescaped* characters, such as `\t` for the tab character. When entering configuration values in the API, use *escaped* characters, such as `\\t` for the tab character.

Query Fields Stage

The Query Fields query pipeline stage defines common Solr query parameters for the edismax query parser *(https://cwiki.apache.org/confluence/display/solr/The+Extended+DisMax+Query+Parser)*. If using a less-than sign (<) with DisMax, it must be escaped using a backslash. An alternative to this stage is the *Additional Query Parameters stage*.

Configuration

When entering configuration values in the UI, use *unescaped* characters, such as `\t` for the tab character. When entering configuration values in the API, use *escaped* characters, such as `\\t` for the tab character.

REST Query Stage

The REST Query pipeline stage is available in Fusion versions 3.0 and later.

In Fusion version 3.0 and earlier, this pipeline stage was called the Query RPC Stage. RPC is the acronym for Remote Procedure Calls.

The notation to send text as payload in **Request entity (as string)** field is `<request.q>`.

Configuration

When entering configuration values in the UI, use *unescaped* characters, such as `\t` for the tab character. When entering configuration values in the API, use *escaped* characters, such as `\\t` for the tab character.

Recommend Items for Item Stage

The Recommend Items for Item query pipeline stage uses signals about users' item choices to recommend related items based on a specific item. Relationships between items can be based on different criteria, such as click patterns, people who bought this also bought that, percentage match of document tags, and so on.

Given an item ID, this stage performs a secondary query to the `COLLECTION_NAME_items_for_item_recommendations` collection to find related items, then retrieves those items from the main collection.

This pipeline stage uses items-for-item recommendations that have been precomputed by the *BPR Recommender*.

See also *Items-for-item Recommendations* to learn how to configure this recommender type and fetch recommendations.

Prerequisites

Enable recommendations:

Before creating a Recommend Items for Item stage, enable recommendations.

- **Fusion UI** – In Collections Manager, click the settings icon next to the collection > **Enable Recommendations**.
- **Using the REST API** – Use this command to enable recommendations:

```
curl -u USERNAME:PASSWORD -X PUT
http://FUSION_HOST:FUSION_PORT/api/v1/collections/COLLECTION_NAME/features/recom
mendations -H 'content-type: application/json' -d '{"enabled":true}'
```

> When you enable recommendations, Fusion creates a query pipeline that already contains this stage, and that is configured for boosting. The query pipeline is `COLLECTION_NAME_items_for_item_recommendations`.

Using live signals

The **Estimate Recent Results** option uses live signals to augment items-for-user recommendations with real-time recommendations.

When this is enabled, Fusion first looks up items (from previously-generated recommendations) that are similar to the new items. If there are none then it looks up similar users (who also interacted with that item) to get a list of recommendations based on the new items. It then combines those new recommendations with the job-based recommendations already generated for that user (if any), to generate a final list of recommendations.

Configuration

> When entering configuration values in the UI, use *unescaped* characters, such as `\t` for the tab character. When entering configuration values in the API, use *escaped* characters, such as `\\t` for the tab character.

Recommend Items for User Stage

The Recommend Items for User query pipeline stage uses signals about item choices to recommend other similar items for a specific user. Personalization for the user can be based on the user's search history, browsing history, or purchase history, and so on.

This pipeline stage uses items-for-user recommendations that have been precomputed by the *BPR Recommender*.

See also *Items-for-user Recommendations* to learn how to configure this recommender type and fetch recommendations.

Prerequisites

Enable recommendations:

Before creating a Recommend Items for Item stage, enable recommendations.

- **In the Fusion UI** – With Query Workbench open, click **Settings > Enable Recommendations**.
- **Using the REST API** – Use this command to enable recommendations:

```
curl -u USERNAME:PASSWORD -X PUT
http://FUSION_HOST:FUSION_PORT/api/v1/collections/COLLECTION_NAME/features/recom
mendations -H 'content-type: application/json' -d '{"enabled":true}'
```

When you enable recommendations, Fusion creates a query pipeline that already contains this stage, and that is configured for boosting. The query pipeline is `COLLECTION_NAME_items_for_user_recommendations`.

Using live signals

The **Estimate Recent Results** option uses live signals to augment items-for-user recommendations with real-time recommendations.

When this is enabled, Fusion first looks up items from previously generated recommendations that are similar to the new items. If there are none then it looks up similar users who also interacted with that item to get a list of recommendations based on the new items. It then combines those new recommendations with the job-based recommendations already generated for that user, to generate a final list of recommendations.

Configuration

When entering configuration values in the UI, use *unescaped* characters, such as `\t` for the tab character. When entering configuration values in the API, use *escaped* characters, such as `\\t` for the tab character.

Recommend Similar Queries Stage

This stage provides query-to-query recommendations.

The stage works in the following way:

1. Perform an exact match of the incoming query to queries stored in the similar queries collection. Return results if a match is found.
2. If **Match non-exact matches** is enabled, the pipeline attempts a normal query and returns the results.
3. If Enable POS tagging is checked, the stage will perform POS tagging on the incoming query and use only the noun parts for the non-exact match.

The spaCy model for part-of-speech (POS) tagging no longer ships with fusion. It needs to be deployed as a Seldon container using the *Create Seldon Model deployment job*.

Configuration

When entering configuration values in the UI, use *unescaped* characters, such as `\t` for the tab character. When entering configuration values in the API, use *escaped* characters, such as `\\t` for the tab character.

Response Document Exclusion Stage

This is a *response rewriting* stage that drops all documents that match all of the specified rules (Boolean AND). If some field has multiple values then at least one value must match against the specified pattern. No further processing is done on any matching documents, and they will not be returned in the final set of results. All non-matching documents are passed to the next stage in the pipeline.

This stage must be used *after* the *Solr Query stage* in your pipeline.

Configuration

When entering configuration values in the UI, use *unescaped* characters, such as `\t` for the tab character. When entering configuration values in the API, use *escaped* characters, such as `\\t` for the tab character.

Response Pairwise Swap Stage

This is a *response rewriting* stage that helps "de-bias" results by swapping the search results at any two positions, such as positions 1 and 2, positions 3 and 4, and so on. This stage must be used *after* the *Solr Query stage* in your pipeline.

This stage is especially useful when combined with *experiments*, where a small percentage of traffic is given swapped results to minimize position bias when using click signals for ranking. The *Response Shuffle stage* provides an another method for minimizing this bias.

Configuration

When entering configuration values in the UI, use *unescaped* characters, such as `\t` for the tab character. When entering configuration values in the API, use *escaped* characters, such as `\\t` for the tab character.

Response Shuffle Stage

This is a *response rewriting* stage that helps "de-bias" results by shuffling the top N results randomly. Since most users click on result #1 before #2, this stage is helpful for distributing clicks more evenly among the top N results. This stage must be used *after* the *Solr Query stage* in your pipeline.

This stage is especially useful when combined with *experiments*, where a small percentage of traffic is given shuffled results to minimize position bias when using click signals for ranking. The *Response Pairwise Swap stage* provides an another method for minimizing this bias.

Configuration

When entering configuration values in the UI, use *unescaped* characters, such as `\t` for the tab character. When entering configuration values in the API, use *escaped* characters, such as `\\t` for the tab character.

Return Query Parameters Stage

The Return Query Parameters query pipeline stage is used to return query parameters as a pipeline response. The stage takes no additional properties beyond id, type, label, skip, and condition.

Configuration

When entering configuration values in the UI, use *unescaped* characters, such as `\t` for the tab character. When entering configuration values in the API, use *escaped* characters, such as `\\t` for the tab character.

Rollup Aggregation Stage

The Rollup Aggregation query pipeline stage (previously called the Rollup Aggregator stage) is used to roll up Solr results from a context variable. In the REST API, this stage type is named `rollup-rec-aggr`.

This stage reads the Solr results (`SolrResponse.class`) from the context and rolls up over a single field product a list of unique IDs and also aggregates the weights (any numeric field in Solr) for those IDs using any of the statistical aggregation functions available. The result from aggregation is saved back in the context and can be used later in a Parameterized Boosting stage.

Configuration

When entering configuration values in the UI, use *unescaped* characters, such as `\t` for the tab character. When entering configuration values in the API, use *escaped* characters, such as `\\t` for the tab character.

Run Query Pipeline Stage

Configuration

When entering configuration values in the UI, use *unescaped* characters, such as `\t` for the tab character. When entering configuration values in the API, use *escaped* characters, such as `\\t` for the tab character.

Security Trimming Stage

The Security Trimming query pipeline stage restricts query results according to the user ID. While indexing the content, the Fusion connectors service stores security ACL metadata associated with the crawled items and indexes them as fields. The Security Trimming stage matches this information against the ID of the user running the search query.

This stage supports *asynchronous processing*.

Configuration

When entering configuration values in the UI, use *unescaped* characters, such as `\t` for the tab character. When entering configuration values in the API, use *escaped* characters, such as `\\t` for the tab character.

Solr MoreLikeThis Stage

This stage uses the content of the current document to query for similar documents, using *Solr's MoreLikeThis component*.

This stage provides content-based recommendations. For collaborative recommendations, use the *Recommend Items for Item stage*.

See *Recommendations and Boosting* for more information.

Tips

- The incoming query must include an `id` field in order to get recommendations from this stage. The stage returns documents similar to the one specified by this field.
- Since these secondary queries tend to be large, this stage can impact search performance. You can improve performance by first *clustering your documents*, then configuring this stage to query a specific document cluster instead of all documents.

Configuration

When entering configuration values in the UI, use *unescaped* characters, such as `\t` for the tab character. When entering configuration values in the API, use *escaped* characters, such as `\\t` for the tab character.

Solr Query Stage

The Solr Query stage transforms the Fusion query pipeline *Request* object into a Solr query and sends it to Solr.

Configuration

When entering configuration values in the UI, use *unescaped* characters, such as `\t` for the tab character. When entering configuration values in the API, use *escaped* characters, such as `\\t` for the tab character.

Solr Subquery Stage

The Solr Subquery query pipeline stage is used to query external collections. The Solr Subquery stage runs a query and adds the results to the pipeline Request object. The query must return results in JSON format, so that they can be added to the query pipeline.

This stage supports *asynchronous processing*.

Configuration

When entering configuration values in the UI, use *unescaped* characters, such as `\t` for the tab character. When entering configuration values in the API, use *escaped* characters, such as `\\t` for the tab character.

Text Tagger Stage

This stage uses the *SolrTextTagger* handler to identify known entities in the query by searching the `COLLECTION_NAME_query_rewrite` *collection*. See *Manage Collections in the Fusion UI* for more information.

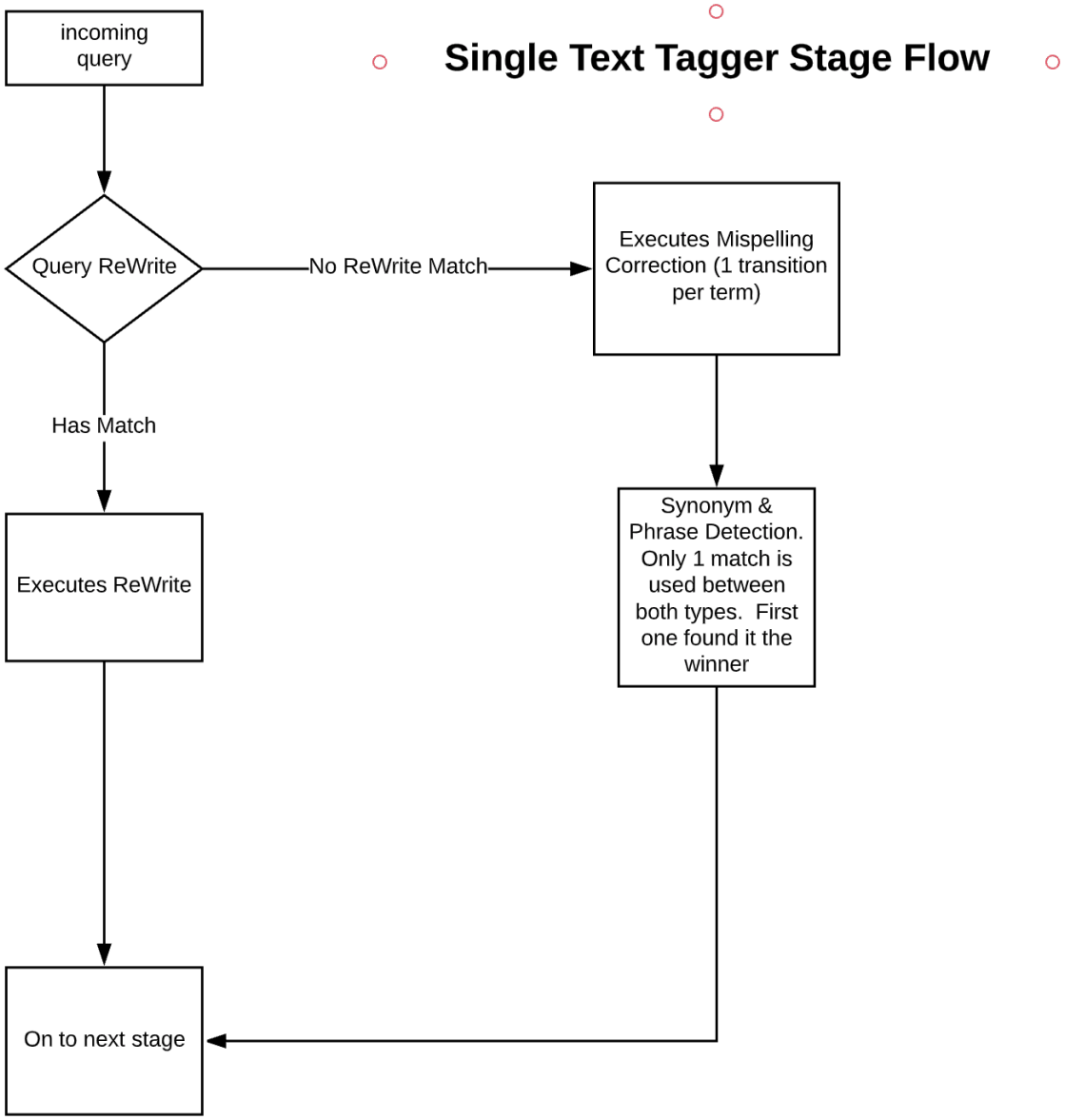
> For Fusion 5.x.x organizations that do *not* have a Predictive Merchandiser license, the Solr Text Tagger handler also searches the `COLLECTION_NAME_query_rewrite_staging` collection in the case of the Fusion query rewriting *Simulator*).

The purpose of the search is to perform *query rewriting* using matches from the following items:

- *Spelling corrections*
- *Phrase boosts*
- *Head/Tail Analysis (Underperforming query improvements)*
- *Synonym expansions*

When the query rewrite entails boosting, the boosting is applied later in the pipeline, during the *Solr Query stage*.

The below diagram shows the process flow for the Text Tagger Stage:



> The underlying SolrTextTagger currently only supports *single-shard collections*. Fusion users should ensure their `COLLECTION_NAME_query_rewrite` collection, or whatever collection the Text Tagger stage is configured to use, is single-sharded before enabling this stage.

Although this stage is available without a Fusion license, it is only effective after running Fusion jobs or creating Fusion rules. See *Query Rewriting* for details.

Configuration

When entering configuration values in the UI, use *unescaped* characters, such as `\t` for the tab character. When entering configuration values in the API, use *escaped* characters, such as `\\t` for the tab character.

Index Pipeline Stages Reference

Index Pipeline stages are used to create and modify *PipelineDocument objects*. Use the *Index Workbench* to configure stages in a pipeline and preview the results.

Call Data Model Pipeline Index Stage

Data models simplify the process of getting started with Fusion by providing pre-configured objects to reduce the effort spent on basic starting tasks. This helps keep documents consistent between datasources and intuitive to the object's type.

The Call Data Model Pipeline index stage is used to create a link between your document type (person, email, file, etc.) and your documents.

This index stage must be placed *after* the ***Data Model Mapping index stage***.

More information

- *How to Configure a Data Model*
- *Data Models*
- *Data Models API reference*
- *Data Model Mapping Index Stage reference*

Configuration

When entering configuration values in the UI, use *unescaped* characters, such as `\t` for the tab character. When entering configuration values in the API, use *escaped* characters, such as `\\t` for the tab character.

Call Pipeline Index Stage

The Call Pipeline index stage provides a means of composing index pipelines.

Be aware that the processing does not return to the calling pipeline automatically.

Configuration

When entering configuration values in the UI, use *unescaped* characters, such as `\t` for the tab character. When entering configuration values in the API, use *escaped* characters, such as `\\t` for the tab character.

Data Model Mapping Index Stage

Data models simplify the process of getting started with Fusion by providing pre-configured objects to reduce the effort spent on basic starting tasks. This helps keep documents consistent between datasources and intuitive to the object's type.

The Data Model Mapping index stage is used to map the field values found in your documents to the data model fields.

	This index stage must be placed *before* the ***Call Data Model Pipeline index stage***.

More information

- *How to Configure a Data Model*
- *Data Models*
- *Data Models API reference*
- *Call Data Model Pipeline Index Stage reference*

Configuration

	When entering configuration values in the UI, use *unescaped* characters, such as `\t` for the tab character. When entering configuration values in the API, use *escaped* characters, such as `\\t` for the tab character.

Date Parsing Index Stage

The Date Parsing Stage (previously called the Date Parser stage) is an index pipeline stage that performs parsing and normalization of date/time data in document fields which uses the Fusion DateUtils library. The resulting date/time information is available both as an timestamp in UTC time zone as well as a local date/time in the original local time zone.

The time zone name, offset and the epoch time are stored in separate fields, too. Additionally the formatted dates can be split into their components, and each component added to separate document fields.

Note that this stage works only with data that consists solely of the date/time information, i.e. it will not work correctly if dates are a part of a larger piece of text.

Timestamp splitting options

Splitting options help in processing timestamp information without resorting to scripting - e.g. in order to index day of week information it is more convenient and faster to split the timestamp in this stage, and then just discard other components that are not needed (using a field mapping stage), rather than using a JavaScript stage to parse and split the timestamp manually.

Please note that time zone name and time zone offset, as well as epoch time, are always added as separate fields regardless of the splitting options. E.g. for a field named `test` these values will be added as fields `tz.test`, `tz_offset.test`, and `epoch.test`.

The option `splitLocal` splits the timestamp in its original timezone, while the option `splitUTC` first converts the timestamp to UTC and then splits it. The resulting date and time components are stored in fields that follow patterns <part>.local.<fieldName> and <part>.utc.<fieldName> respectively.

The following parts are extracted and added to the document:

- **year.** year component
- **month.** month in year, from 1 to 12
- **day.** day in month, from 1 to 31
- **yday.** day in year, from 1 to 356
- **weekday.** day of week, 1 being Monday and 7 being Sunday
- **week.** week in year, from 1 to 52. Note: in the standard ISO8601 week algorithm, the first week of the year is that in which at least 4 days are in the year. As a result of this definition, day 1 of the first week may be in the previous year, which will be indicated by weekyear. The opposite is also true - last day of the last week may be in the next year, and weekyear will show the next year.
- **weekyear.** year corresponding to the week value. This can be either the current year or previous one, or the next one.
- **hour.** hour in day, from 0 to 23
- **min.** minute in hour, from 0 to 59
- **sec.** second in minute, from 0 to 59
- **ms.** millisecond in second, from 0 to 999

Example: given this normalized timestamp in the original timezone `2015-01-01 00:00:00.000 Europe/Warsaw` in a field `test`, the corresponding normalized UTC timestamp will be `2014-12-31T23:00:00.00Z`.

Example: `splitLocal` parsing

The following table shows the additional fields added to a document as the result of applying `splitLocal` parsing to the contents a field named `test` which contains the value `2015-01-01 00:00:00.000 Europe/Warsaw`:

Field name	value
tz.test	Europe/Warsaw
tz_offset.test	+01:00
epoch.test	1420066800000

Example: `splitUTC` parsing

The following table shows the additional fields added to a document as the result of applying `splitUTC` parsing to the contents a field named `test` which contains the value `2015-01-01 00:00:00.000 Europe/Warsaw`:

Field name	value
tz.test	Europe/Warsaw
tz_offset.test	+01:00
epoch.test	1420066800000
year.utc.test	2014
year.local.test	2015
month.utc.test	12
month.local.test	1
day.utc.test	31
day.local.test	1
yday.utc.test	365
yday.local.test	1
weekday.utc.test	3
weekday.local.test	4
week.utc.test	1
week.local.test	1
weekyear.utc.test	2015
weekyear.local.test	2015
hour.utc.test	23
hour.local.test	0
min.utc.test	0
min.local.test	0
sec.utc.test	0
sec.local.test	0
ms.utc.test	0

Field name	value
ms.local.test	0

Note: The following:

- weekday is different. UTC day of week is Wednesday, and local day of week is already Thursday.
- yday in UTC points to the last day of the year, while it is the first day of the year in local time zone, similarly with day.
- week and weekyear are the same in both cases. because according to the ISO 8601 definition all days of this week belong to year 2015 so it does not matter whether it is Wednesday or Thursday.

Configuration

Adding a New Format

If no new format is added, a default list of formats will be used. The default formats are listed in the table below:

Pattern	Notes
`EEE yyyy-MM-dd HH:mm:ss.SSS zzz`	
`yyyy-MM-dd'T'HH:mm:ss.SSSZ`	With numeric +-HHmm timezone at the end
`yyyy-MM-dd'T'HH:mm:ss.SSSZZ`	With numeric +-HH:mm timezone at the end
`yyyy-MM-dd'T'HH:mm:ss.SSSz`	With symbolic XXX timezone at the end
`yyyy-MM-dd'T'HH:mm:ssz`	With symbolic XXX timezone at the end
`yyyy-MM-dd'T'HH:mm:ssZ`	With offset
`EEE MMM d HH:mm:ss z yyyy`	
`EEE MMM d HH:mm:ss Z yyyy`	
`EEE MMM d HH:mm:ss z yyyy`	
`EEE MMM d HH:mm:ss.SSS z yyyy`	
`EEE, dd MMM yyyy HH:mm:ss zzz`	
`EEEE, dd-MMM-yy HH:mm:ss zzz`	
`yyyy-MM-dd HH:mm:ss Z`	
`yyyy-MM-dd HH:mm:ss ZZ`	
`yyyy-MM-dd HH:mm:ss z`	
`yyyy-MM-dd HH:mm:ss.SSS Z`	
`yyyy-MM-dd HH:mm:ss.SSS ZZ`	
`yyyy-MM-dd HH:mm:ss.SSS z`	
`yyyy-MM-dd HH:mm:ss zzz`	With full time zone (e.g. America/New_York)
`yyyy-MM-dd'T'HH:mm:ss'GMT'Z`	With literal "GMT" and offset
`yyyy-MM-dd'T'HH:mm:ss.SSS'GMT'Z`	With literal "GMT" and offset
`yyyy-MM-dd'T'HH:mm:ss'UTC'Z`	With literal "UTC" and offset
`yyyy-MM-dd'T'HH:mm:ss.SSS'UTC'Z`	With literal "UTC" and offset
`yyyy-MM-dd HH:mm:ss 'UTC'Z`	

Pattern	Notes
`yyyy-MM-dd HH:mm:ss.SSS 'UTC'Z`	
`yyyy-MM-dd HH:mm:ss 'GMT'Z`	
`yyyy-MM-dd HH:mm:ss.SSS 'GMT'Z`	
`dd-MM-yyyy HH:mm:ss zzz`	

The list of accepted symbols for constructing a new format are listed below:

Symbol	Meaning	Example
`G`	era	`AD`
`C`	century of era (>=0)	`20`
`Y`	year of era (>=0)	`1996`
`x`	weekyear	`1996`
`w`	week of weekyear	`27`
`e`	day of week	`2`
`E`	day of week	`Tuesday` `Tue`
`y`	year	`1996`
`D`	day of year	`189`
`M`	month of year	`July` `Jul` `07`
`d`	day of month	`10`
`a`	halfday of day	`PM`
`K`	hour of halfday (0~11)	`0`
`h`	clockhour of halfday (1~12)	`12`
`H`	hour of day (0~23)	`0`
`k`	clockhour of day (1~24)	`24`
`m`	minute of hour	`30`
`s`	second of minute	`55`
`S`	fraction of second	`978`
`z`	time zone	`Pacific Standard Time`; `PST`
`Z`	time zone offset/id	`-0800` `-08:00` `America/Los_Angeles`
`'`	escape for text	
`''`	single quote	`'`

The number of symbols used determine the format of the output.

For text output, if 4 or more symbols are used, the full form is used. `EEEE` would output `Tuesday`, while `E` outputs `Mon`.

For numeric outputs, the amount of symbols used is the same as the minimum digits used for representation.

If the number is smaller, then it will be padded with zeroes. `HH` would be `09`, while `H` is `9`.

For the year, the representation for year and weekyear is different. For example, if the count of `y` is two, the year would be the two digit, zero-based year of the century.

For the month, if there are three or more symbols, then the text is used. For example, `MM` would output `03`, `MMM` outputs `Mar`, and `MMMM` outputs `March`.

For the zone, `Z` outputs the offset without a colon, `ZZ` outputs the offset with a colon, and `ZZZ` or more outputs the zone id.

For the zone name, time zone names `'z'` cannot be parsed.

Any characters in the imput that are not in the range of `['a'…'z']` and `['A'…'Z']` will be treated like quoted text. So, characters like `','` or `'-'` will appear in the output time text even if they are not wrapped in single quotes.

A new configuration can be made using a combination of the symbols and regular characters. For example, the pattern `E MM/dd/yyyy HH:mm:ss.SSS` would output the formatted date `Mon 01/16/2017 10:53:32.939`.

	When entering configuration values in the UI, use *unescaped* characters, such as `\t` for the tab character. When entering configuration values in the API, use *escaped* characters, such as `\\t` for the tab character.

Detect Language Index Stage

The Detect Language index stage (called the Language Detection stage in versions earlier than 3.0) operates over one or more fields in the Pipeline Document. The contents of each field are analyzed using the Language Detection Library for Java *(https://github.com/optimaize/language-detector)*, which is an open-source project hosted on GitHub. The analyzer returns the ID of the language which best matches the contents of that field, if any. These IDs can be returned as an annotation on the Pipeline Document context, or as annotation on each field analyzed.

The language identification algorithm breaks the text in each source field into ngrams and compares them to sets of ngrams compiled from all the different language versions of the Wikipedia. This library will only produce reasonable results for document fields which are comparable in length, vocabulary, and style to the known texts compiled from the Wikipedia. Caveats are discussed below.

If a positive language identification is made, that information is added to the Pipeline Document according to the choice of configuration property "Output Type". If the language annotation is added to the PipelineDocument context object, the name of the context key string is specified by configuration property "Output Key". For Output Type configuration property "Document", per-field language annotations are added to the document using a parallel naming convention where the name of the language identification field starts with the name of the analyzed field and has an additional suffix string, default value "_lang". For example, if a document contains fields named "plot_summary_txt" and "user_reviews_txt" to be analyzed, if the software can detect the language, it will add fields "plot_summary_txt_lang" and "user_reviews_txt_lang".

There is also an option to allow detection of multiple languages. This can be achieved by setting "Return all detected languages and their confidence scores." to true. In this case, the detected languages will be either set as document fields in a form of "Field Name_Document Postfix.Language:Confidence", or as a field with name "Output Key" in the Context having a dictionary of following form "{ "language":"probability" }" as a value. Example Document fields could look like this: "plot_summary_txt_lang.pl_: [0.99]", "plot_summary_txt_lang.en_: [0.99]" when languages pl and en would be detected.

Languages

The Language Detection Library for Java has build-in profiles for many languages *(https://github.com/optimaize/language-detector/blob/master/README.md#70-built-in-language-profiles)*. If there is a set of Wikipedia entries written in a language, it is likely that the Language Detection Library can identify texts written in this language.

Caveats

This library should produce reasonable results on document fields which are comparable in length, vocabulary, and style to the known texts compiled from the Wikipedia.

The documentation lists the following challenges *(https://github.com/optimaize/language-detector/blob/master/README.md#challenges)*:

- This software does not work as well when the input text to analyze is short, or unclean. For example tweets.
- When a text is written in multiple languages, the default algorithm of this software is not appropriate. You can try to split the text (by sentence or paragraph) and detect the individual parts. Running the language guesser on the whole text will just tell you the language that is most dominant, in the best case.
- This software cannot handle it well when the input text is in none of the expected (and supported) languages.
- Detection of unwanted languages (for example the stage might detect some language that is not even used in the input data because of some language similarities). By default, the stage uses a full array of available languages for detection (List here *(https://github.com/optimaize/language-detector)*). If one wants to only use selected languages, this can be configured

Configuration

	When entering configuration values in the UI, use *unescaped* characters, such as `\t` for the tab character. When entering configuration values in the API, use *escaped* characters, such as `\\t` for the tab character.

Encode into Milvus Index Stage

The Encode into Milvus index stage invokes a machine learning model to encode a field and store it into *Milvus*.

Troubleshooting

Update a Milvus collection

Update and delete semantics are not supported on individual items in a Milvus collection. We recommend that you use a blue/green set of collections: reindex to apply updates to an empty Milvus collection, then swap between collections to apply the updates.

Configuration

	When entering configuration values in the UI, use *unescaped* characters, such as `\t` for the tab character. When entering configuration values in the API, use *escaped* characters, such as `\\t` for the tab character.

Exclude Documents Index Stage

The Exclude Documents stage drops all documents that match all of the specified rules (Boolean AND). If some field has multiple values then at least one value must match against the specified pattern. No further processing is done on any matching documents, thus they will not be indexed into a Fusion collection. All non-matching documents are passed to the next stage in the pipeline. Rules are defined using regular expression field matching.

Examples

Give the "simple-exclude" pipeline a stage that excludes certain document types

```
curl -u USERNAME:PASSWORD -X POST -H "Content-type: application/json"
'http://FUSION_HOST:FUSION_PORT/api/index-pipelines' -d ' { "id" : "simple-
exclude", "stages" : [ { "type" : "exclude-doc", "matchRules" : [ { "field" :
"document_type", "pattern" : "(xls|xlsx|xlst|doc|docx)" }] }] }'
```

Send a text document through the "simple-exclude" pipeline

```
curl -u USERNAME:PASSWORD 'http://FUSION_HOST:FUSION_PORT/api/index-
pipelines/simple-exclude/collections/logs/index?simulate=true&echo=true' -H
'Content-type: application/json' -d ' { "document_type": "txt" }'
```

The response is document metadata, indicating the document passed the stage:

```
[ { "id" : "93da43ff-4218-4f24-a690-23b530926104", "fields" : [ { "name" :
"document_type", "value" : "txt", "metadata" : { }, "annotations" : [ ] } ] } ]
```

Send an XLS document through the "simple-exclude" pipeline

```
curl -u USERNAME:PASSWORD 'http://FUSION_HOST:FUSION_PORT/api/index-
pipelines/simple-exclude/collections/logs/index?simulate=true&echo=true' -H
'Content-type: application/json' -d ' { "document_type": "xls" }'
```

The empty response indicates the document was dropped:

```
[ ]
```

Configuration

	When entering configuration values in the UI, use *unescaped* characters, such as `\t` for the tab character. When entering configuration values in the API, use *escaped* characters, such as `\\t` for the tab character.

Exclusion Filter Index Stage

The Exclusion Filter index stage is used to remove fields or documents that match items in a pre-defined exclusion list.

There are two ways to supply an exclusion list:

- Upload a file containing a newline-separated list, using the *Blob Store*. When configuring the index stage, reference the list by its blob name in the `location` property (**Exclusion List URI** in the Fusion UI).
- When configuring the index stage, enter an array of values for exclusion in the `excludeValues` property (**Exclusion List** in the Fusion UI).

The Exclusion Filter stage can be configured using one or both of these methods; Fusion combines them into one list. If `regexPattern` is configured, the pattern is applied to the field before the result is compared to the combined list.

By default, any matching *field* is excluded from indexing. To exclude the whole document, set `skipDocument` to "true" (**Skip Document** in the Fusion UI).

Uploading an exclusion list

Before you can configure the `location` property, you must upload one or more exclusion lists to Fusion using the *Blob Store API*.

Fusion comes with an example exclusion list at `https://FUSION_HOST:FUSION_PORT/data/nlp/excludes/excludes.txt`. Here is an example of how to upload this file using `curl`, where `USERNAME:PASSWORD` are the credentials for an admin-level user:

```
curl -u USERNAME:PASSWORD -X PUT --data-binary @data/nlp/excludes/excludes.txt -H
'Content-type: text/plain' http://FUSION_HOST:FUSION_PORT/api/blobs/excludes.txt
```

Example

Use an exclusion list for entities found in the author field:

```
{ "type" : "exclusion-filter", "id" : "iw", "filters" : [ { "sourceField" :
"author_s", "location" : "excludes.txt", "caseSensitive" : false } ], "skip" :
false } ] }
```

Configuration

	When entering configuration values in the UI, use *unescaped* characters, such as `\t` for the tab character. When entering configuration values in the API, use *escaped* characters, such as `\\t` for the tab character.

Field Mapping Index Stage

A Field Mapping stage is used to do customized mapping of the fields in an Index Pipeline document to fields in the Solr schema.

For examples of how to use this stage in the Fusion UI, see *Part 2* of the *Getting Started* tutorial.

Field Mapping Stage Properties

A Field Mapping stage specification consists of three things:

- a unique ID
- a set of mapping rules that specify operations applied to named fields as a triple: `{ source, target, operation }`.
- a set of rules called "unmapped" rules which specify operations applied to fields whose name does not match any of the mapping rules, also a triple `{ source, target, operation }`.

Mapping Rules and Unmapped Rules

Each rule has the following properties:

Property	Description
source	The name of the source field. This will be the name of the field in the Pipeline document that should be mapped to another field. Java regular expressions can be used in the source field by surrounding the regular expression with forward slashes ('/'). For example, `/(.*)text(.*)/` is a valid expression that will find field names in the incoming document that contain the string 'text' between any number of preceding or succeeding characters. If a regular expression is not used, the value supplied for the source will be treated as a literal field name and will be matched ignoring the case (for example, "text" will match "tExt" or "Text", etc.).
target	The name of the target field. If the value for the `source` was a regular expression, then this can also be a regular expression. It can also contain substitutions using references to capture groups (using Java's Matcher.replaceAll *(http://docs.oracle.com/javase/8/docs/api/java/util/regex/Matcher.html#replaceAll-java.lang.String-)*). Otherwise, the source field name will be simply substituted by the value of target according to the operation rules described below.
operation	What to do with the field during mapping. Several options are available: • **copy.** Content contained in fields matching `source` will be copied to `target`. • **move.** Content contained in fields matching `source` will be moved to `target` (it may also help to think of this as the field name being replaced by the value of `target`). • **delete.** Content contained in fields matching `source` will be dropped from the document and not indexed. In this case, the `target` can be null or not defined at all. • **add.** The literal value of `target` will be added to the `source` if `source` is a regular expression. If `source` is not a regular expression, `target` will be added as a new field. • **set.** The literal value of `target` will be set as the new value of `source` if `source` is a regular expression. If `source` is not a regular expression, `target` will be set as a new field. • **keep.** Content contained in fields matching `source` will be retained and unchanged, and the fields will be added to a list of known fields and they will not be affected by however the `renameUnknown` rule has been set.

Order of operations
Field mapping rules are applied in the order in which they are defined within each operation *type*. Operation types are applied in this order:

1. **Field Retention** - `keep` or `delete`
2. **Field Value Updates** - `add` or `set`
3. **Field Translations** - `copy` or `move`

As a result, `keep` and `delete` rules are always applied before `add`, `set`, `copy`, or `move` rules. Likewise, `add` and `set` rules are always applied before `copy` or `move` rules.

In some cases, you may wish to delete fields after they are processed by other operations. To accomplish this, you can add another Field Mapping stage which deletes the fields.

Field Mapping Behavior

The field mapping rules are applied in a specific order.

1. A copy of the Pipeline document is prepared. All further operations are applied to this copy.
2. The rules are traversed only once, in the order of their declaration in the `mapping` property. This means it is possible to do multiple operations on a field. However, note that if fields are moved (renamed), further operations should reference the new field name.
3. Before each rule is evaluated, the current list of field names is prepared and sorted in alphabetic ascending order.
4. The current rule is applied to field values for each matching name from the list of names prepared in step 3. New field names resulting from the current rule do not effect the snapshot list of field names; in order for a rule to be applied to a new field name, it will be included in a later round of the evaluation cycle.
5. The process is repeated for each rule, and a list of matching source fields is noted.
6. If the document contains any fields that were not affected by any mapping rule, the `renameUnknown` option is applied if it has been set to true.
7. Finally, the resulting transformed document is returned to the next stage of the index pipeline.

Examples

Map several fields:

```
{ "id": "mapping-text", "type": "field-mapping", "mappings": [ { "operation":
"move", "source": "plaintextcontent", "target": "body" }, { "operation": "add",
"source": "content-length", "target": "fileSize" }, { "operation": "move",
"source": "/file(.*)/", "target": "lastModified" }, { "operation": "delete",
"source": "last-printed" }, { "operation": "copy", "source": "mimetype", "target":
"content_type" } ], "unmapped": { "source": "/(.*)/", "target": "$1_ss",
"operation": "move" }, "skip" : false }
```

Set the `urlX` field based on the value of the `employee_id` field:

```
{ "id": "set-field", "type": "field-mapping", "mappings": [ { "operation": "set",
"source": "urlX", "target": "https://mydomain.com/<employee_id>" } ], "skip" :
false }
```

Configuration

	When entering configuration values in the UI, use *unescaped* characters, such as `\t` for the tab character. When entering configuration values in the API, use *escaped* characters, such as `\\t` for the tab character.

Field Parser Index Stage

Parse the contents of a field using a pre-defined parser.

Configuration

	When entering configuration values in the UI, use *unescaped* characters, such as `\t` for the tab character. When entering configuration values in the API, use *escaped* characters, such as `\\t` for the tab character.

Filter Short Fields Index Stage

A Filter Short Fields index stage removes short field values from a pipeline document according to a set of filters where each filter specifies a field name and a minimum length. All field values less than the specified length will be removed from the document.

For the REST API, this stage type is "filter-short-fields".

Example Stage Specification

Remove entities in the `author_s` field that are less than 3 characters:

```
{ "type" : "filter-short-fields", "id" : "short-stage", "filters" : [ {
"sourceField" : "author_s", "length" : 3 } ], "skip" : false }
```

Configuration

	When entering configuration values in the UI, use *unescaped* characters, such as `\t` for the tab character. When entering configuration values in the API, use *escaped* characters, such as `\\t` for the tab character.

Find and Replace Index Stage

The Find and Replace stage uses exact string matching to standardize or remove a set of specific passages, phrases, or words from a document field.

This stage is configured with one or more rules. Each rule specifies:

- One or more text strings to search for matching text (phrases or words)
- A *single* text string that is the replacement value

> When the replacement string is empty, the system deletes the matched text from the document field.

Configure the Find and Replace stage in one of the following ways:

- *Fusion blob store setup*
- *Fusion UI or REST API setup*

Fusion blob store setup

Setting up the stage using Fusion's blob store is referred to as **Find - List - Replace**.

The general procedure for *each* find/list/replace action is:

- Upload a file that contains a list of text passages to be standardized into the *Fusion Blob Store*.
- Each file:
  - Contains one line for each text passage
  - Has a `.lst` suffix

The required rule properties to standardize the text are the:

- Document field to scan
- Name of the `.lst` file uploaded to the blob store
  > The `.lst` suffix must be included in the file name designated in the rule.
- Value of the replacement text

> Multiple `.lst` files can be uploaded and used to perform find/replace actions.

Example

One example could be to remove the country information and replace it with the phrase: **country not displayed for security purposes** to make the data anonymous for reporting.

- The document field to scan is **country**.
- The name of the `.lst` file in this example is `country.lst`. An excerpt is:

```
Afghanistan Afrique Albania Albanie Alderney Algeria Algérie Allemagne America
Amérique Amériques American Samoa Andorra Andorre Angleterre Anglo-Normandes Angola
Anguilla Antigua and Barbuda Antigua et Barbuda Antilles Antilles Néerlandaises
Arabie Saoudite Argentina Argentine Arménie Aruba
```

- The value of the replacement text is: **country not displayed for security purposes**.

If the text of the passages scanned includes any of the items in the `country.lst` file, it is replaced with the phrase: **country not displayed for security purposes**.

Fusion UI or REST API setup

Setting up the stage using the Fusion UI or REST API is referred to as **Find - Replace**.

The Fusion UI access path is:

- Sign in to Fusion.
- Select the application.
- Select **Indexing > Index Workbench**.
- Select **Add a Stage**.
- Select **Find and Replace** in the Field Transformation section.

- Enter values in the fields as described in the *Configuration section*.

  The REST API uses the parameters specified below.

The required rule properties to standardize the text are the:

- Document field to scan
- List of find/replace pairs

Configuration

When entering configuration values in the UI, use *unescaped* characters, such as `\t` for the tab character. When entering configuration values in the API, use *escaped* characters, such as `\\t` for the tab character.

Format Signals Index Stage

Configuration

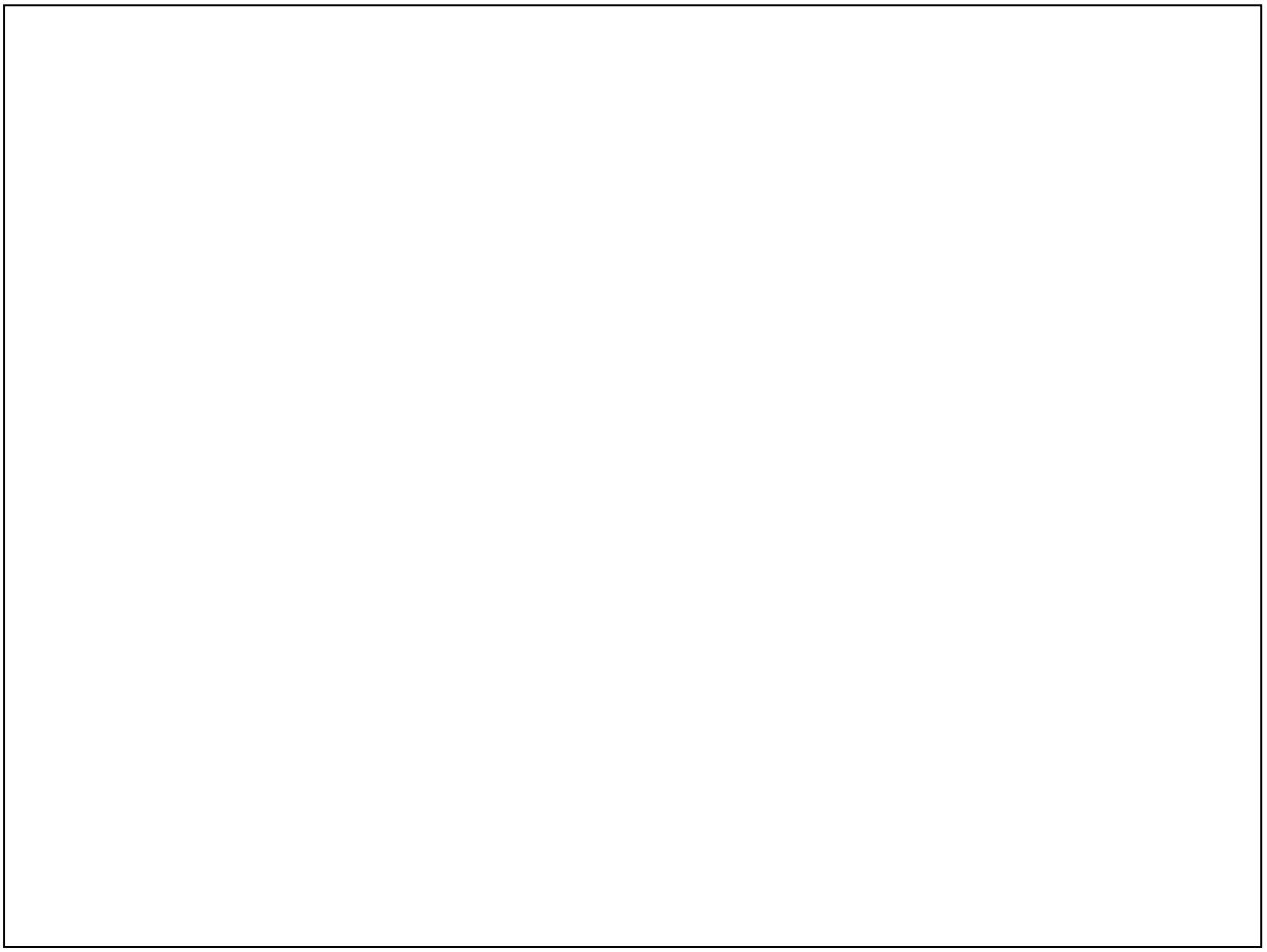
When entering configuration values in the UI, use *unescaped* characters, such as `\t` for the tab character. When entering configuration values in the API, use *escaped* characters, such as `\\t` for the tab character.

Gazetteer Lookup Extraction Index Stage

The Gazetteer Lookup Extraction index stage (called the Gazetteer Lookup Extractor stage in versions earlier than 3.0) uses predefined lists of words and phrases to process specified text fields in a document. A gazetteer is a set of lookup lists over names of people, places, or things. These lookup lists are used to find occurrences of these names in text. The matched items are saved into separate fields on the document for downstream processing.

Gazetteers and OpenNLP Tools

The following video shows how to configure a Gazetteer Lookup Extraction stage in combination with OpenNLP:



Uploading Lookup Lists to Fusion Blob Store

Fusion includes a number of lookup lists in the directory at `https://FUSION_HOST:FUSION_PORT/data/nlp/gazetteer`. To use the supplied lists or a list of your own data, each must list be uploaded to Fusion using the *Blob Store API* in order to make the list contents available to the Gazetteer Lookup Extraction stage.

For example, to identify color names, you would first compile a list of color terms, one entry per line in a text file with suffix `.lst` and then upload that file using the Fusion REST API endpoint `api/blobs/<listfilename>`, as per the following example which uses the `curl` command-line utility, where 'admin' is the name of a user with admin privileges, and 'pass' is that user's password:

```
curl -u USERNAME:PASSWORD -X PUT --data-binary @data/nlp/gazetteer/colors.lst -H
'Content-type: text/plain' http://FUSION_HOST:FUSION_PORT/api/blobs/colors.lst
```

Name Lookup Example

Define a lookup-extractor to identify mentions of certain celebrities in text field `description_t`*:*

```
{ "type" : "lookup-extractor", "id" : "peopleLookup", "rules" : [ { "source" : [
"description_t" ], "target" : "celebrities_ss", "entityTypes" : [ { "name" :
"person_female", "definitions" : [ "person_female.lst" ] } ], "additionalEntities"
: [ { "name" : "players", "definitions" : [ "sharapova", "murray" ] }, { "name" :
"actors", "definitions" : [ "pitt", "jolie" ] } ], "caseSensitive" : false } ],
"skip" : false }
```

Configuration

	When entering configuration values in the UI, use *unescaped* characters, such as `\t` for the tab character. When entering configuration values in the API, use *escaped* characters, such as `\\t` for the tab character.

GeoIP Lookup Stage

This index pipeline stage looks up the geographical location of an IP address.

If the **IP Address Field** (`ipAddrField`) is specified, it will be used first to perform the geolocation lookup. If the lookup is successful, then the **Hostname Field** (`hostnameField`) will be skipped. However, if the lookup failed or the **IP Address Field** is blank, and the **Hostname Field** is specified, then the stage will try to perform the lookup using the hostname.

The default geo database is GeoLite2-City *(https://dev.maxmind.com/geoip/geoip2/geolite2/)* from MaxMind. To use a different database (for which you have a commercial license), you can *upload it to the blob store* (in binary format), then specify its blob ID in the **Geoip Lookup Database Blob ID** (`blobId`) parameter.

Fields added in this stage

When this stage is run, specific fields are added to the document. If the value is not present, the value added is `undefined`.

Field	Definition
`geo_loc_coord`	The latitude and longitude coordinates of the location. For example, 37.751,-97.822.
`geo_postal_code`	The postal code of the location.
`geo_country_name`	The country of the geographic location. For example, United States.
`geo_subdivision_iso`	The subdivision of the country in an ISO code format. For example, the ISO code for the state or province of the geographic location.
`geo_lon`	The longitude of the location. For example, -97.822.
`geo_country_iso`	The two-letter ISO code for the country of the location. For example, US.
`geo_subdivision_name`	The subdivision name of the location in the country. For example, the name of the state or province of the geographic location.
`geo_lat`	The latitude of the location. For example, 37.751.
`geo_city_name`	The city name of the location. For example, Chicago.

Configuration

	When entering configuration values in the UI, use *unescaped* characters, such as `\t` for the tab character. When entering configuration values in the API, use *escaped* characters, such as `\\t` for the tab character.

Include Documents Index Stage

This stage passes documents to the next stage in the pipeline if they match one or more of the specified rules (Boolean OR). If some field has multiple values then at least one value must match against specified pattern. All non-matching documents are dropped. Rules are defined using regular expression field matching.

Examples

Give the "simple-include" pipeline a stage that includes only certain document types

```
curl -u USERNAME:PASSWORD -X POST -H "Content-type: application/json"
'http://FUSION_HOST:FUSION_PORT/api/index-pipelines' -d ' { "id" : "simple-
include", "stages" : [ { "type" : "include-doc", "matchRules" : [ { "field" :
"document_type", "pattern" : "(xls|xlsx|xlst|doc|docx)" }] }] }'
```

Response:

```
{ "id" : "simple-include", "stages" : [ { "type" : "include-doc", "id" : "f701f96b-
780e-4355-9dd3-6e53a89afe3e", "matchRules" : [ { "field" : "document_type",
"pattern" : "(xls|xlsx|xlst|doc|docx)" } ], "type" : "include-doc", "skip" : false,
"label" : "include-doc" } ], "properties" : { } }
```

Send a text document through the "simple-include" pipeline

```
curl -u USERNAME:PASSWORD 'http://FUSION_HOST:FUSION_PORT/api/index-
pipelines/simple-include/collections/logs/index?simulate=true&echo=true' -H
'Content-type: application/json' -d ' { "document_type": "txt" }'
```

The empty response indicates the document was dropped:

```
[ ]
```

Send an XLS document through the pipeline

```
curl -u USERNAME:PASSWORD 'http://FUSION_HOST:FUSION_PORT/api/index-
pipelines/simple-include/collections/logs/index?simulate=true&echo=true' -H
'Content-type: application/json' -d ' { "document_type": "xls" }'
```

The response is document metadata, indicating the document passed the stage:

```
{ "id" : "9e7d1c2e-343a-49de-bc6a-1d1fc25fa93f", "fields" : [ { "name" :
"document_type", "value" : "xls", "metadata" : { }, "annotations" : [ ] } ] } ]
```

Configuration

	When entering configuration values in the UI, use *unescaped* characters, such as `\t` for the tab character. When entering configuration values in the API, use *escaped* characters, such as `\\t` for the tab character.

JDBC Index Stage

The JDBC Index Stage is used to connect to a database, lookup one or more values, and then inject them into the context. The properties for setting stage are identical to the *JDBC Query Stage*, except for the "rows" property, which defaults to **-1** for the index stage (which returns all rows in the database), and defaults to **10** returned rows for the query stage.

	You must first *upload the JDBC driver to Fusion*.

Example

An example of a JDBC Index Stage setup

Upload stage config via POST to Fusion REST API endpoint `api/index-stages/instances`

```
curl -u USERNAME:PASSWORD -X POST -H "Content-Type: application/json" -d '{"id":
"jdbc-index-test","type":"jdbc-index-lookup","driver":"postgresql-9.3-1101-
jdbc4.jar","connection":"jdbc:postgresql:database","username":"user","password":"pa
ssword1","preparedStatement":"select ID as id from DATABASE;"}'
http://FUSION_HOST:FUSION_PORT/api/index-stages/instances
```

Response

```
{ "type" : "jdbc-index-lookup", "id" : "jdbc-index-test", "driver" : "postgresql-
9.3-1101-jdbc4.jar", "connection" : "jdbc:postgresql:database", "username" :
"user", "password" : "password1", "preparedStatement" : "select ID as id from
DATABASE;", "fetchSize" : -1, "join" : true, "rows" : -1, "skip" : false, "label" :
"jdbc-index-lookup", "type" : "jdbc-index-lookup" }
```

Configuration

	When entering configuration values in the UI, use *unescaped* characters, such as `\t` for the tab character. When entering configuration values in the API, use *escaped* characters, such as `\\t` for the tab character.

JavaScript Index Stage

The JavaScript Index stage allows you to write a custom processing logic using JavaScript to manipulate Pipeline Documents and the index pipeline context, which will be compiled by the JDK into Java bytecode that is executed by the Fusion pipeline. The first time that the pipeline is run, Fusion compiles the JavaScript program into Java bytecode using the JDK's JavaScript engine.

> Users who can create or modify code obtain access to the broader Fusion environment. This access can be used to create intentional or unintentional damage to Fusion.

For a JavaScript Index stage, the JavaScript code must return either:

- A single document or array of documents
  or
- The null value or an empty array.
  In the latter case, no further processing is possible, which means that the document will not be indexed or updated. For example, Solr commits have a `null` value and are dropped. For information about how to skip stages when Solr commits are sent, see the *Skip JavaScript stages during Solr commits*.

JavaScript Index Stage Global Variables

JavaScript *(http://en.wikipedia.org/wiki/JavaScript)* is a lightweight scripting language. The JavaScript in a JavaScript stage is standard ECMAScript *(http://en.wikipedia.org/wiki/ECMAScript)*. What a JavaScript program can do depends on the container in which it runs. For a JavaScript Index stage, the container is a Fusion index pipeline. The following global pipeline variables are available:

Name	Type	Description
`doc`	*Pipeline Document*	The contents of each document submitted to the pipeline.
`ctx`	*Context*	A map that stores miscellaneous data created by each stage of the pipeline. > Use the `ctx` variable instead of the deprecated `_context` global variable. The `ctx` variable is used to: • Pass data from one stage to another • Store data that needs to be passed from one custom stage to a later custom stage The data can differ between stages: • If the previous stage changes the data • Based on the configuration of each stage > If the data is modified in one stage, it may cause a later stage to function irregularly.
`collection`	String	The name of the Fusion collection being indexed or queried.
`solrServer`	*BufferingsolrServer*	The Solr server instance that manages the pipeline's default Fusion collection. All indexing and query requests are done by calls to methods on this object. See solrClient *(https://lucene.apache.org/solr/5_2_1/solr-solrj/org/apache/solr/client/solrj/SolrClient.html)* for details
`solrServerFactory`	*solrClusterComponent*	The SolrCluster server used for lookups by collection name which returns a Solr server instance for that collection. For example: `var productsSolr = solrServerFactory.getSolrServer("products");`.

Syntax Variants

JavaScript stages can be written using legacy syntax or function syntax. The key difference between these syntax variants is how the "global variables" are used. While using legacy syntax, these variables are used as global variables. With function syntax, however, these variables are passed as function parameters.

Legacy Syntax

Legacy syntax is used to perform very simple tasks.

```
//do some work... return doc;
```

Function Syntax

Function syntax is used for moderately complex tasks.

```
function (doc) { // do some work ... return doc; }
```

Function syntax is used for the examples in this document.

Advanced Syntax

Advanced syntax is used for complex tasks and when multiple functions are needed.

```
/* globals Java, logger*/ (function () { "use strict"; return function main(doc ,
ctx, collection, solrServer, solrServerFactory) { // do some work ... return doc;
}; })();
```

JavaScript Use

The JavaScript in a JavaScript Index stage must return either a single document or an array of documents. This can be accomplished by either:

- a series of statements where the final statement evaluates to a document or array of documents
- a function that returns a document or an array of documents

All pipeline variables referenced in the body of the JavaScript function are passed in as arguments to the function. For example, in order to access the PipelineDocument in global variable 'doc', the JavaScript function is written as follows:

```
function (doc) { // do some work ... return doc; }
```

The allowed set of function declarations are:

```
function doWork(doc) { ... return doc; } function doWork(doc, ctx) { ... return
doc; } function doWork(doc, ctx, collection) { ... return doc; } function
doWork(doc, ctx, collection, solrServer) { ... return doc; } function doWork(doc,
ctx, collection, solrServer, SolrClusterComponent) { ... return doc; }
```

The order of these arguments is according to the (estimated) frequency of use. The assumption is that most processing only requires access to the document object itself, and the next-most frequent type of processing requires only the document and read-only access of some context parameters. If you need to reference the solrServerFactory global variable, you must use the 5-arg function declaration.

In order to use other functions in your JavaScript program, you can define and use them, as long as the final statement in the program returns a document or documents.

Global variable `logger`

The logs are output to the indexing service logs for custom index stages. Access the Log Viewer and filter on this service to view the information.

The JavaScript Engine Used by Fusion

The JavaScript engine used by Fusion is the Nashorn engine from Oracle. See The Nashorn Java API *(https://docs.oracle.com/javase/8/docs/technotes/guides/scripting/nashorn/api.html)* for details.

Creating and accessing Java types

The following information is taken from Oracle's JavaScript programming guide section 3, Using Java From Scripts *(https://docs.oracle.com/javase/8/docs/technotes/guides/scripting/prog_guide/javascript.html#A1147207)*.

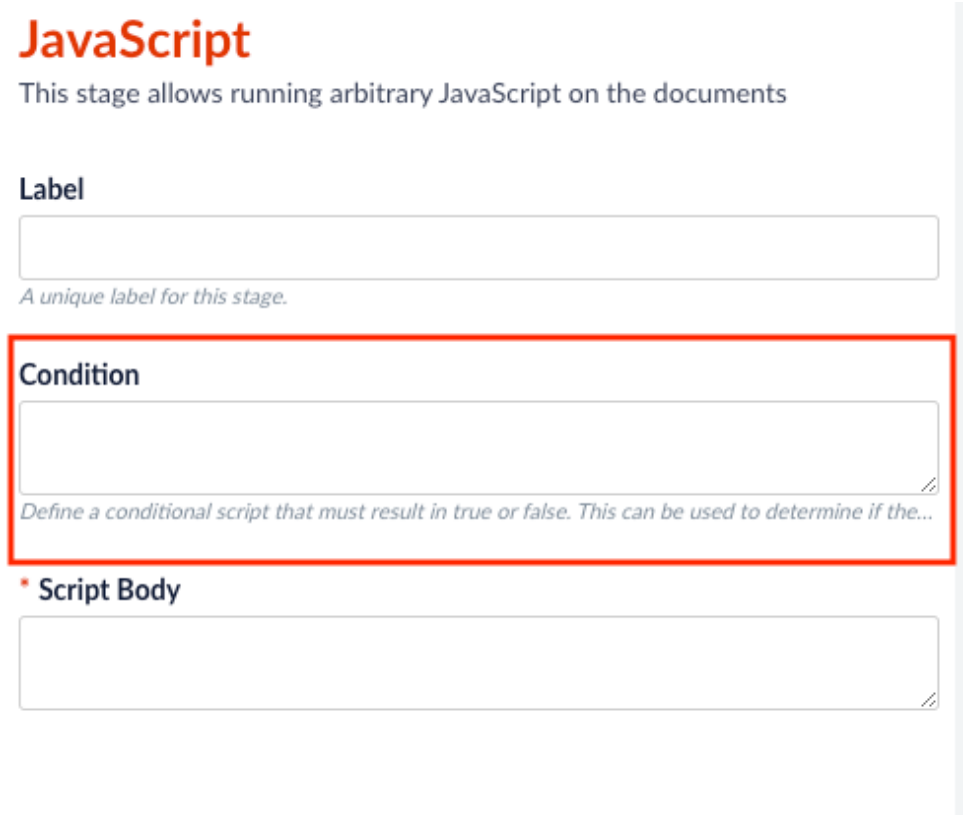
To create script objects that access and reference Java types from Javascript use the `Java.type()` function:

```
var ArrayList = Java.type("java.util.ArrayList"); var a = new ArrayList;
```

Examples

Set the condition field

The JavaScript Index stage lets you define a condition to trigger the script body.



The condition field is evaluated as either true or false. Do not precede with `if` or include `;` at the end of the line.

- Works:
  - `doc.hasField("title_s") === true`
  - `doc.hasField("title_s") === false`
  - `doc.hasField("title_s")`
- Does not work:

- `if doc.hasField("title_s") === false;`

Skip JavaScript stages during Solr commits

When JavaScript stages contain data manipulation such as values in the `conditions` field, errors may be generated and Solr commits are dropped because the value is `null`. To skip those stages when a Solr commit is sent through the pipeline, add the following condition to those stages: `doc.getCommands().size()==0`.

Add a field to a document

```
function (doc) { doc.addField('some-new-field', 'some-value'); return doc; }
```

Join two fields

The following example conjoins separate latitude and longitude fields into a single geo-coordinate field, whose field name follows Solr schema conventions and ends in "_p". It also removes the original latitude and longitude fields from the document.

```
function (doc) { var value = ""; if (doc.hasField("myGeo_Lat") &&
doc.hasField("myGeo_Long")) { value = doc.getFirstFieldValue("myGeo_Lat") + "," +
doc.getFirstFieldValue("myGeo_Long"); doc.addField("myGeo_p", value);
doc.removeFields("myGeo_Lat"); doc.removeFields("myGeo_Long");
logger.debug("conjoined Lat, Long: " + value); } return doc; }
```

Return an array of documents

```
function (doc) { var subjects = doc.getFieldValues("subjects"); var id =
doc.getId(); var newDocs = []; for (i = 0; i < subjects.size(); i++) { var pd = new
com.lucidworks.apollo.common.pipeline.PipelineDocument(id+'-'+i );
pd.addField('subject', subjects.get(i)); newDocs.push( pd ); } return newDocs; }
```

Parse a JSON-escaped string into a JSON object

While it is simpler to use a JSON Parsing index stage, the following code example shows you how to parse a JSON-escaped string representation into a JSON object.

This code parses a JSON object into an array of attributes, and then find the attribute "tags" which has as its value a list of strings. Each item in the list is added to a multi-valued document field named "tag_ss".

```
function (doc, ctx) { /** * Returned body is a JSON string. Example: { "telescope":
"Hubble", "location": "low earth orbit", "primary_mirror_size": "2.4m" "tags":
["launched 1990","repaired 1993","repaired 1997","repaired 1997","repaired
1999","repaired 2002","repaired 2009"] } */ for (var key in myData) { var obj =
myData[key]; if (obj == "tags") { logger.debug("extracting tags from tag field {}",
JSON.stringify(obj)) for (var i = 0; i < obj.length; i++) { var tag = obj[i][0];
doc.addField("tag_ss", tag); } } } return doc; }
```

Do a lookup on another Fusion collection

```
function doWork(doc, ctx, collection, solrServer, solrServerFactory) { var sku = doc.getFirstFieldValue("sku"); if (!doc.hasField("mentions")) { var mentions = "" var productsSolr = solrServerFactory.getSolrServer("products"); if( productsSolr != null ){ var q = "sku:"+sku; var query = new org.apache.solr.client.solrj.SolrQuery(); query.setRows(100); query.setQuery(q); var res = productsSolr.query(query); mentions = res.getResults().size(); doc.addField("mentions",mentions); } } return doc; }
```

Reject a document

If the function returns `null` or an empty array, it will not be indexed or updated into Fusion.

```
function doWork(doc) { if (!doc.hasField("required_field")) { return null; } return doc; }
```

Drop a document by ID

```
function(doc) { var id = doc.getId(); if (id !== null) { var pattern = "https://www.mydomain.com/links/contact/?"; // 0 means the pattern was found so drop the doc return (id.indexOf(pattern) == 0) ? null : doc; } return doc; }
```

Format Date to Solr Date

```
// For example: // From: 26/Mar/2015:14:38:48 -0700 // To: 2015-03-26T14:38:48Z (Solr format) function(doc) { if (doc.getId() !== null) { var inboundPattern = "dd/MMM/yyyy':'HH:mm:ss Z"; // modify this to match the format of the inbound date var solrDatePattern = "yyyy-MM-dd'T'HH:mm:ss'Z'"; // leave this alone var dateFieldName = "apachelogtime"; // change this to your date field name var solrFormatter = new java.text.SimpleDateFormat(solrDatePattern); var apacheParser = new java.text.SimpleDateFormat(inboundPattern); var dateString = doc.getFirstFieldValue(dateFieldName); logger.info("**** dateString: " + dateString); var inboundDate = apacheParser.parse(dateString); logger.info("**** inboundDate: " + inboundDate.toString()); var solrDate = solrFormatter.format(inboundDate); logger.info("**** solrDate: " + solrDate.toString()); doc.setField(dateFieldName, solrDate.toString()); } return doc; }
```

Replace whitespace and newlines

```
function(doc) { if (doc.getId() !== null) { var fields = ["col1", "col2", "col3"]; for (i = 0; i < fields.length; i++ ) { var field = fields[i]; var value = doc.getFirstFieldValue(field); logger.info("BEFORE: Field " + field + ": *" + value + "*"); if (value != null) { value = value.replace(/^\s+/, ""); // remove leading whitespace logger.info("AFTER: Field " + field + ": *" + value + "*"); value = value.replace(/\s+$/, ""); // remove trailing whitespace logger.info("AFTER: Field " + field + ": *" + value + "*"); value = value.replace(/\s+/g, " "); // multiple whitespace to one space logger.info("AFTER: Field " + field + ": *" + value + "*"); doc.setField(field, value); } } } return doc; }
```

Split the values in a field

```
//Split On a delimiter. In this case, a newline function(doc){ if (doc.getId() !== null) { var fromField = "company2_ss"; var toField = "company2_ss"; var delimiter = "\n"; var oldList = doc.getFieldValues(fromField); var values = []; // parse the entries one at a time doc.removeFields(toField); // clear out the target field for (i = 0; i < oldList.size(); i++) { values[i] = oldList.get(i); // get the list of strings split by the delimiter newList = values[i].split(delimiter); for(j = 0; j < newList.length; j++ ){ doc.addField(toField, newList[j]); } } } return doc; }
```

Prevent global variables in JavaScript

Variable declared using the `var` keyword

If a variable is declared using the `var` keyword, the JavaScript interpreter processes the value sequentially.

In this example, the values for `var i = 0` are logged in order as `0`, `1`, `2`, `3`, `4`, etc.

```
var queries = ["cat","the cat", "the cat in the hat","the cat in the hat is back"] for(var i = 0; i < queries.length; i++){ logger.info("query {} is '{}'",i,queries[i]) }
```

Variable not declared using the `var` keyword

If a variable is not declared using the `var` keyword, the JavaScript interpreter moves the declaration of variable and functions to the top of the declared (global) scope. Because Fusion pipeline stages execute in a multi-threaded environment, these global (shared) variables make the stages not thread-safe.

For more detailed information, see Hoisting *(https://developer.mozilla.org/en-US/docs/Glossary/Hoisting#:~:text=JavaScript%20Hoisting%20refers%20to%20the,defined%20in%20the%20ECMAScript%20specification)*.

```
var queries = ["cat","the cat", "the cat in the hat","the cat in the hat is back"] for(i = 0; i < queries.length; i++){ logger.info("query {} is '{}'",i,queries[i]) }
```

Issues and errors that may occur include:

For multi-threaded environments, the value of `i` may not proceed sequentially from 0 to 4 as the loop is processed. Instead, values may be logged based on the execution state of the other pipeline requests. For example, `0`, `1`, `3`, `1`, `2`, etc., which logs the values as `"cat", "the cat", "the cat in the hat is back", "the cat", "the cat in the hat"`.

However, if only one thread is incrementing the `i` variable, the values proceed sequentially (`0`, `1`, `2`, `3`, `4`, etc.)

If the `queries` array varies in length from document to document, the loop may generate an `ArrayIndexOutOfBounds` exception for a Java array or an `undefined` error for a JavaScript array.

Threads may not log all four queries.

Setting the `"use strict"` directive

Setting the `"use strict"` directive tells the JavaScript engine to require non-global declarations of all functions and variables.

The following example demonstrates how to create a copy of a `PipelineDocument` and return both the original and the copy to the pipeline for processing.

```
/* globals Java, logger*/ // This is an optional line that informs the syntax checker about known global objects. This avoids excessive flagging of what it perceives as errors. (function () { //Defines a top-level scope "use strict"; // This directs the interpreter to require `var` for all variable declarations to help ensure thread safety. Any global scope variables generate compile errors. /*** * NOTE: All code within the outer scope, (everything from other than the `main` function returned by the `function` declared on line two) is only run when the JavaScript engine JIT-compiles the script. Place comments, static declarations, functions, and initialization code here to improve readability and performance. */ var PipelineDocument = Java.type('com.lucidworks.apollo.common.pipeline.PipelineDocument'); var ArrayList = Java.type("java.util.ArrayList") /** * Take a PipelineDocument like the one passed to this stage and clone it. Then give it a new id. */ function clonePipelineDoc(pipelineDoc, id) { var clone = new PipelineDocument(pipelineDoc); if (id) { clone.setId(id); } logIfDebug("Cloned document with id '{}' and gave the clone id '{}'",pipelineDoc.getId(),id) return clone; }; var isDebug = false // This setting is optional and determines if log messages are turned off or on. It works in conjunction with the `logIfDebug` function. function logIfDebug(m){if(isDebug && m)logger.info(m, Array.prototype.slice.call(arguments).slice(1));} return function main(doc , ctx, collection, solrServer, solrServerFactory) { //This returns the function to be called by Fusion when the stage executes, and guarantees Fusion will always call this function. When a stage does not use advanced syntax, but declares multiple top-level functions, Fusion may not be able to determine which function to call. Duplicate doc and give it an ID that is the reverse of the original. var doc2 = clonePipelineDoc(doc, doc.getId().split("").reverse().join("")) var list = new ArrayList() list.add(doc) list.add(doc2) //Return both docs to the downstream stages. return listOfDocs; }; })();
```

Additional resources

- View our training on JavaScript in Fusion *(https://academy.lucidworks.com/javascript-in-fusion)*.
- *Fusion Pipeline Javadocs*

Configuration

	When entering configuration values in the UI, use *unescaped* characters, such as `\t` for the tab character. When entering configuration values in the API, use *escaped* characters, such as `\\t` for the tab character.

Logging Index Stage

The Logging Index stage prints messages to the Connectors log file. The default location is: `https://FUSION_HOST:FUSION_PORT/var/log/api/fusion-indexing.log`.

The verbosity of this message is controlled by the property `detailed`. When detailed logging is true the current PipelineDocument object is pretty-printed to the Connectors log file.

In a production environment logging stages should be configured with property `skip` set to `true`, if possible. Use of detailed logging may impact performance.

Configuration

	When entering configuration values in the UI, use *unescaped* characters, such as `\t` for the tab character. When entering configuration values in the API, use *escaped* characters, such as `\\t` for the tab character.

Machine Learning Index Stage

The Fusion machine learning indexing stage uses a trained machine learning model to analyze a field or fields of a *PipelineDocument* and stores the results of analysis in a new field of either the PipelineDocument or *Context* object.

In order to use the Machine Learning Stage, you must train a machine learning model. For more information on machine learning in Fusion, see:

- *Machine Learning*
- *Develop and Deploy a Machine Learning Model*

Configuration

	When entering configuration values in the UI, use *unescaped* characters, such as `\t` for the tab character. When entering configuration values in the API, use *escaped* characters, such as `\\t` for the tab character.

Managed JavaScript Index Stage

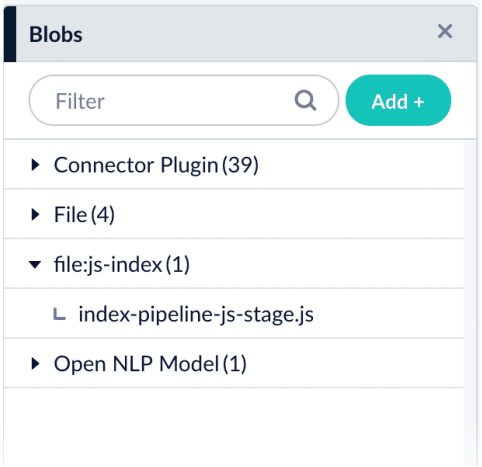
This stage runs a JavaScript transformation on documents using a script from the *blob store*. The script must have the blob type `file:js-index`.

Uploading a JavaScript blob

In Fusion 5.4, JavaScript blobs must be uploaded using the *Blob Store API*, like this:

```
curl -u USERNAME:PASSWORD -X PUT -H 'Content-Type: text/javascript' --data-binary
@index-pipeline-js-stage.js
http://FUSION_HOST:FUSION_PORT/api/apps/APP_NAME/blobs/index-pipeline-js-stage.js?
resourceType=file:js-index
```

Open the *blob manager* to verify that the new blob is present:



Referencing a JavaScript blob

Use any of the following formats for the `ref`/**Script Reference** configuration field:

```
index-pipeline-js-stage.js
```

```
blob:index-pipeline-js-stage.js
```

```
urn:x-fusion:blob:index-pipeline-js-stage.js
```

The examples above use `index-pipeline-js-stage.js` as the blob ID.

Edit JavaScript

For Fusion 5.4.x and later, select **Edit** in the **Script Reference** field to edit JavaScript directly in the Fusion UI.

Configuration

	When entering configuration values in the UI, use *unescaped* characters, such as `\t` for the tab character. When entering configuration values in the API, use *escaped* characters, such as `\\t` for the tab character.

NLP Annotator Index Stage

The NLP Annotator index stage performs *Natural Language Processing* tasks.

	This stage is deprecated as of Fusion 5.2.0. in favor of SpaCy and Seldon Core functionality. It is best practice to use the *Machine Learning Index Stage* instead.

You can choose from different NLP libraries, either OpenNLP *(https://opennlp.apache.org/)* or the JohnSnow Lab *(https://nlp.johnsnowlabs.com/)* library, which runs on Spark.

	Only the pre-trained NER model is supported. If choosing an NER model, download **NerDLModel** instead of NerCRFModel.

The NLP Annotator supports the following tasks:

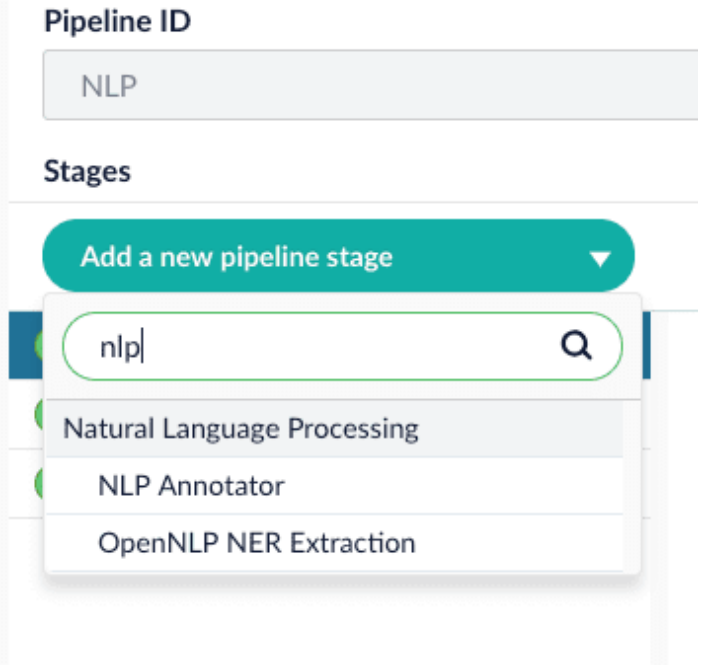
- If choosing JohnSnow Lab (recommended for large dataset processing):
  - NER (Name Entity Recognition)

    Fusion uses the deep learning pre-trained NER model that JohnSnowLab provides. Currently, the pre-trained extraction model covers the following name entities:
    - ORG (organization)
    - PER (person)
    - LOC (location)

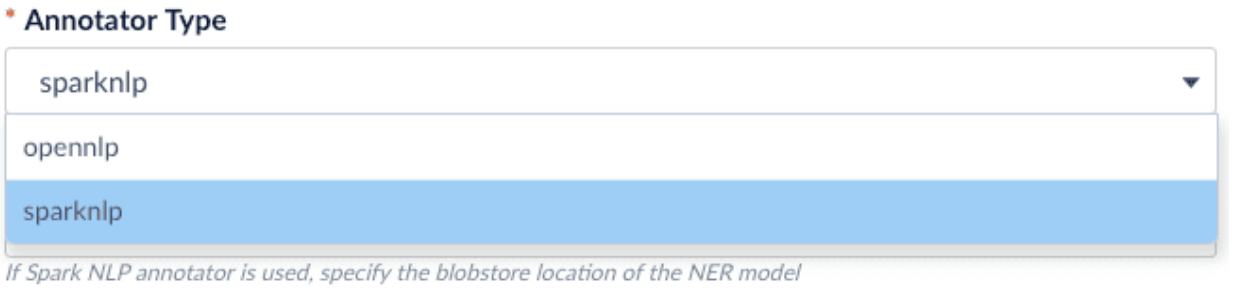
    This means that there are the only three types of entities Fusion will recognize from the source field.
  - Sentence detection
  - POS(Part of Speech) Tagging
- If choosing OpenNLP:
  - NER
  - Sentence detection
  - POS Tagging
  - Shallow Parsing (Chunking)

Example of how to use NLP Annotator Index stage:

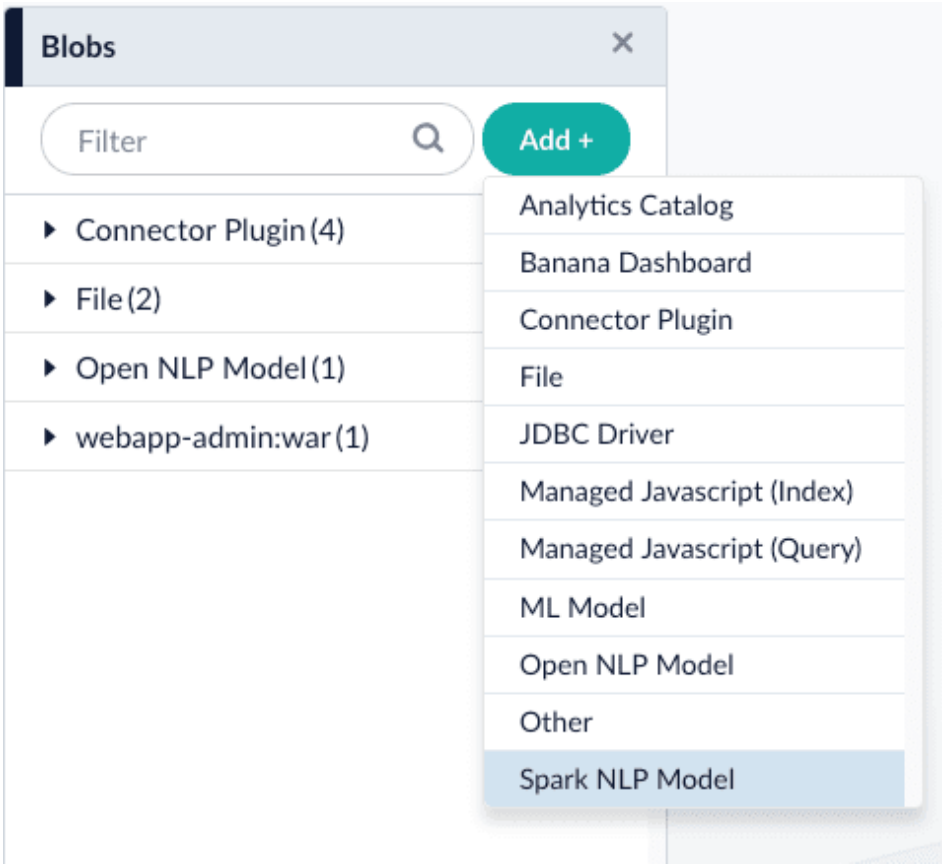
1. Add NLP Annotator index stage.



2. Choose the annotator type (OpenNLP or SparkNLP).

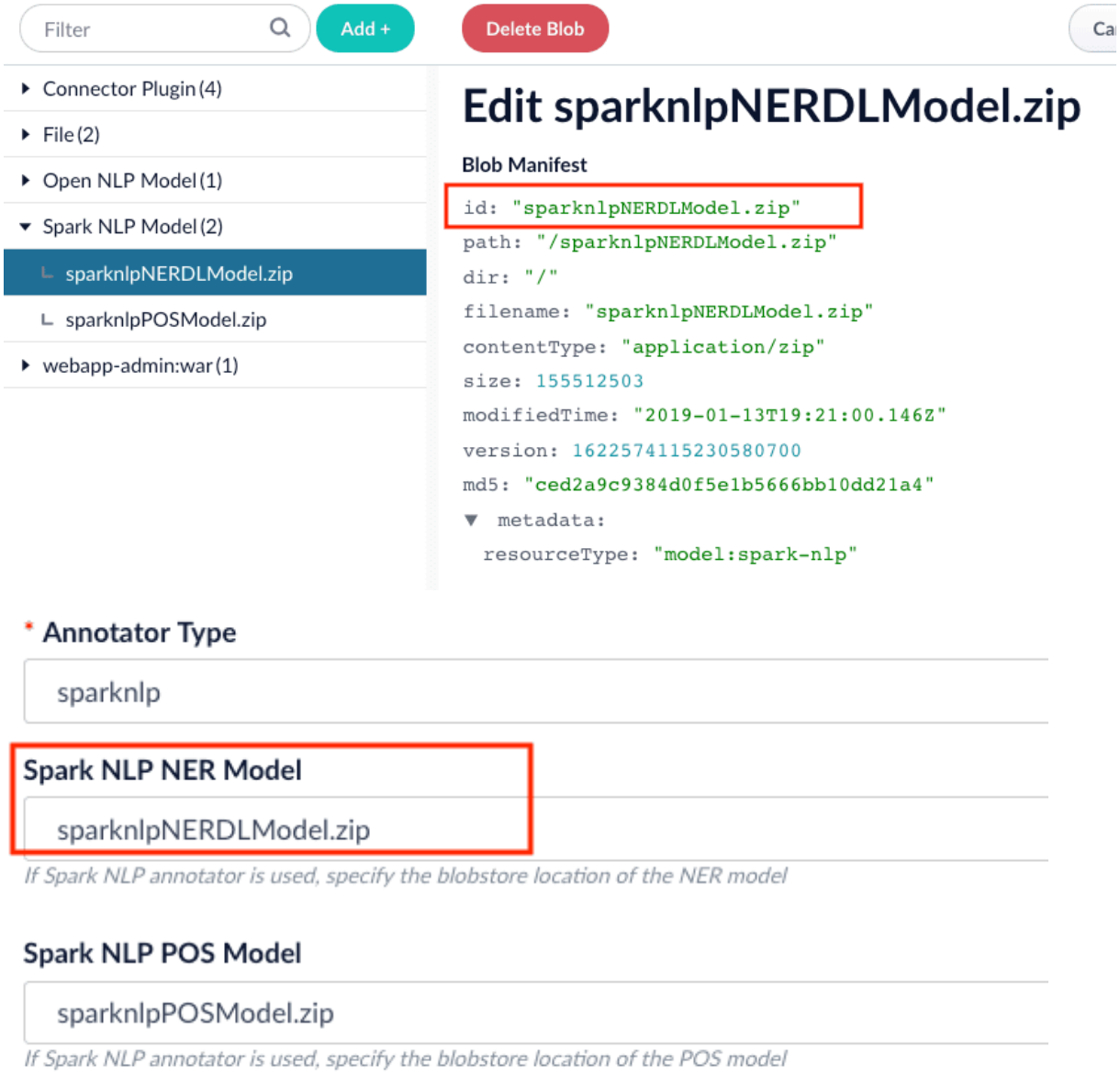


If you select the `sparknlp` model, you need to download and install one or more models: .. Download the models at https://github.com/JohnSnowLabs/spark-nlp#models *(https://github.com/JohnSnowLabs/spark-nlp#models)*. .. Rename the downloaded models to something easy to identify, then upload them to Fusion's *blob store*.



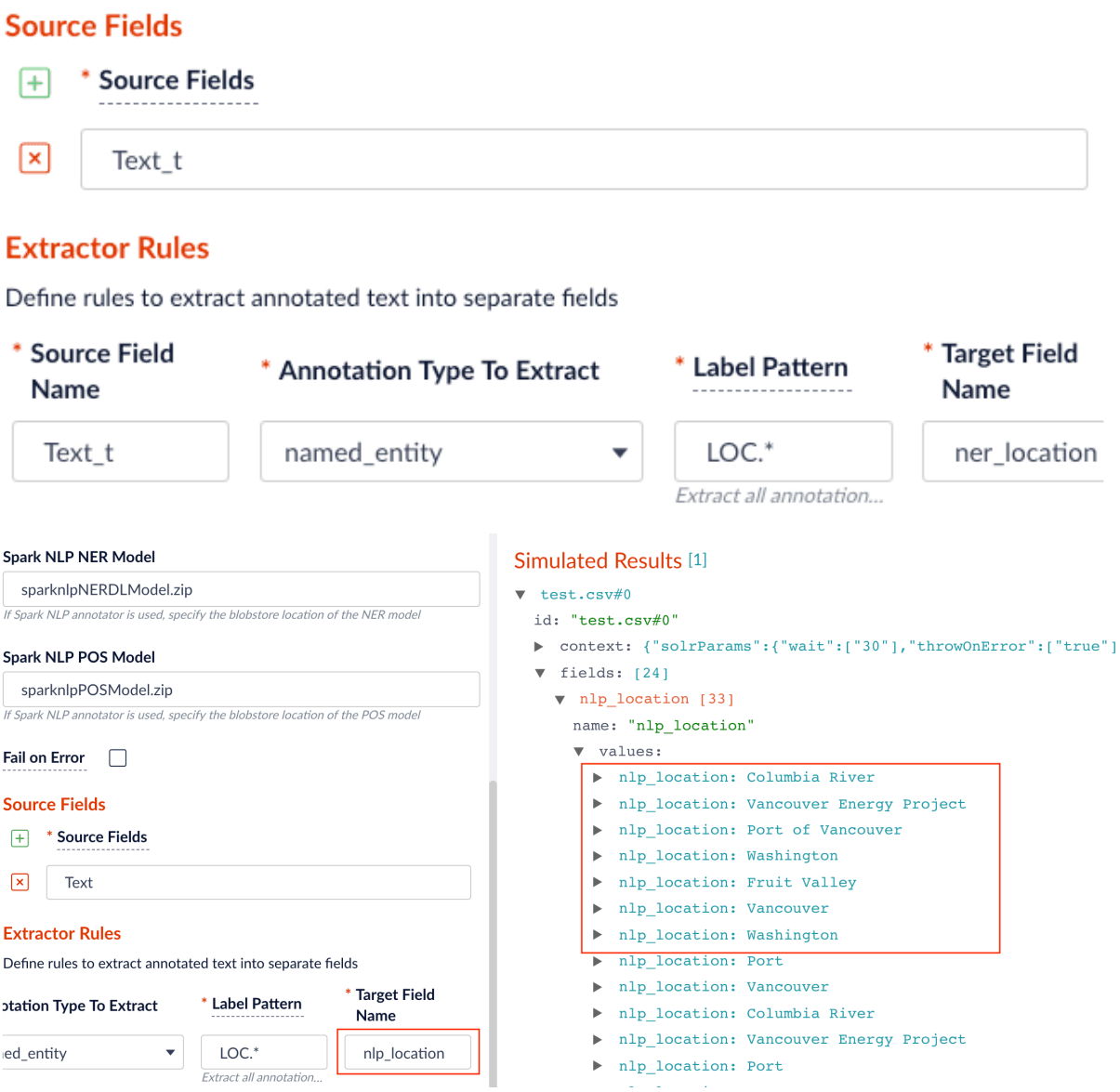
+

3. Configure the index pipeline stage:
   a. Specify the model to use (fill the box with `model id` in the blob store).



b. Specify the source, label pattern, and target (destination) fields:

- source field: the raw text with name entities to be extracted.
- label pattern: regex pattern that matches the NER/POS labels: for example, `PER.` **will match extracted name entities with label** `PERSON`**, while** `NN.` will match tagged nouns.
- target field: the outcome extraction/tagging and so on.



Configuration

	When entering configuration values in the UI, use *unescaped* characters, such as `\t` for the tab character. When entering configuration values in the API, use *escaped* characters, such as `\\t` for the tab character.

OpenNLP NER Extraction Index Stage

Named Entity Recognition (NER) is the task of finding the names of persons, organizations, locations, and/or things in a passage of free text. The OpenNLP NER Extraction index stage (previously called the OpenNLP NER Extractor stage) uses a set of rules to find named entities in a field in the Pipeline Document (the "source") and populates a new field (the "target") with these entities.

	This stage is deprecated as of Fusion 5.2.0. in favor of SpaCy and Seldon Core functionality. SpaCy offers better accuracy and detects more entities than OpenNLP.

This stage uses Apache OpenNLP *(http://opennlp.apache.org/)* project's Named Entity Recognition tool *(http://opennlp.apache.org/docs/#tools.namefind.recognition)* (the Name Finder tool). The OpenNLP documentation states:

> The Name Finder tool can detect named entities and numbers in text. To be able to detect entities the Name Finder needs a model. The model is dependent on the language and entity type it was trained for. The OpenNLP projects offers a number of pre-trained name finder models which are trained on various freely available corpora. They can be downloaded at our model download page. To find names in raw text the text must be segmented into tokens and sentences.

Fusion 5.x.x contains a common set of NER models for English that include sentence, token, and part-of-speech models. These models are:

Model	Purpose
`nlp/models/en-sent.bin`	Sentence model to detect sentences
`nlp/models/en-token.bin`	Tokenizer model for tokenization of sentences
`nlp/models/en-ner-date.bin`	Date name finder model
`nlp/models/en-ner-location.bin`	Location name finder model
`nlp/models/en-ner-money.bin`	Money name finder model
`nlp/models/en-ner-organization.bin`	Organization name finder model
`nlp/models/en-ner-percentage.bin`	Percentage name finder model
`nlp/models/en-ner-person.bin`	Person name finder model
`nlp/models/en-ner-time.bin`	Time name finder model

	See OpenNLP 1.5 series *(http://opennlp.sourceforge.net/models-1.5/)* for additional pre-trained OpenNLP models.

To use these models, upload to Fusion using the *Fusion Blob Store service*. Here is an example of how to upload the sentence model file using the `curl` command-line utility, where "admin" is the name of a user with admin privileges, and "pass" is the password:

```
curl -u USERNAME:PASSWORD -X PUT --data-binary @data/nlp/models/en-sent.bin -H
'Content-type: application/octet-stream'
http://FUSION_HOST:FUSION_PORT/api/blobs/en-sent.bin
```

See *Natural Language Processing* for more information.

Example Specification

Specification of a stage which extracts names of people and places from field named 'body':

```
{ "type":"nlp-extractor", "id":"iqtr", "rules":[ { "source":[ "body_t" ],
"target":"organizations", "writeMode":"append",
"sentenceModelLocation":"nlp/models/en-sent.bin",
"tokenizerModelLocation":"nlp/models/en-token.bin", "entityTypes":[ {
"name":"organization", "definition":"nlp/models/en-ner-organization.bin" } ] }, {
"source":[ "body_t" ], "target":"persons", "writeMode":"append",
"sentenceModelLocation":"nlp/models/en-sent.bin",
"tokenizerModelLocation":"nlp/models/en-token.bin", "entityTypes":[ {
"name":"person", "definition":"nlp/models/en-ner-person.bin" } ] } ], "type":"nlp-
extractor", "skip":false, "label":"Extract Entities", "licensed":true,
"secretSourceStageId":"iqtr" }
```


Configuration

	When entering configuration values in the UI, use *unescaped* characters, such as `\t` for the tab character. When entering configuration values in the API, use *escaped* characters, such as `\\t` for the tab character.

REST Query Index Stage

The REST Query index stage performs remote procedure calls to external services, merges the response from that service with a document being processed by the pipeline, and makes calls to the external system for each document. In Fusion version 3.0 and earlier, the stage is called the Query REST Remote Procedure Calls (RPC) Client stage.

This stage can call external systems in any of the following ways:

- any HTTP or HTTPS request: `<http-protocol>://<path>`
- a Solr request: `solr://COLLECTION_NAME/…`
- a Fusion service request: `extservice://SERVICE_NAME/<path>`

The call syntax allows the use of document field variables. For example, variable `${docField}` will be replaced with the value of the field named `docField` for the document being processed.

Several types of responses are also supported:

- POJOs
- JSON
- XML

Call Parameters

In the Fusion UI, select **include** to display the fields for the call parameters (`params` in the REST API). Parameters are specified as key-value pairs:

Endpoint URI / `uri`	A fully-qualified service URI. This can be an HTTP call, a Solr request, or a Fusion service call * any HTTP or HTTPS request: `<http-protocol>://<path>` * a Solr request: `solr://COLLECTION_NAME/…` * a Fusion service request: `extservice://SERVICE_NAME/<path>`
Call Method / `method`	The method to use for the RPC call. Supported methods are `GET`, `POST`, `PUT`, and `DELETE`.
Query parameters / `queryParams`	Query parameters to be passed with the request.
Request protocol headers / `headers`	Request protocol headers, such as `"Content-Type" : "application/json"`.
Request entity (as string) / `entity`	The request body. This parameter can only be configured via the REST API; *see the example below*.

The `params` can also take variable substitution expressions. Variables are expressed as `${fieldName}`, where `fieldName` is a name of the current document's field (or `id` for the document's id). Only the first value of a multi-valued field is used for substitution, and this value is treated as a string.

For example, a `queryParams` could be constructed as follows:

```
"params" : { "uri" : "solr://collection1/select", "method" : "GET", "queryParams" : { "q" : "{!field f=a_txt v=$prodName}", "prodName" : "${prodName}" }
```

In this example, the variable `${prodName}` will be replaced with the string value of the 'prodName' field in the current document being processed. If the current document contains "iPhone 6" as the value of 'prodName', the resulting query in this example will be a fielded search `q=a_txt:iPhone 6`.

Mapping Rules

The **mappingRules** property takes the following, specified as key-value pairs:

- path. an XPath expression *(http://www.w3schools.com/xml/xpath_syntax.asp)*. It is assumed that this expression always returns a node list, and each returned node is processed separately and its converted value is added to a multi-valued field.
- target. the name of the target field in the current document where the value(s) will be stored.
- append. if true, values extracted from the RPC response will be appended to the target multi-valued field. When this is false, the default, existing values in the target field will be discarded and replaced with the values from the RPC response.
- xml. if true, the extracted DOM nodes will be separately serialized to XML and the resulting XML-formatted text will be added to the target field. When this is false, the default, the extracted DOM nodes will be flattened and converted to a list of fields. Field names in this case will correspond to XML element names, dot-separated, and element attributes will be represented as fields with @attributeName suffix.

Examples

Create a REST Query stage to merge results from another Solr system:

```
{ "type" : "indexing-rpc", "id" : "BasicSolrCall", "mappingRules" : [ { "path" :
"//result/doc[1]/str[@name='foo_s']/text()", "target" : "foo_s", "append" : true,
"xml" : false }, { "path" : "//result/doc/arr[@name='a_txt']/*", "target" :
"doc_txt", "append" : true, "xml" : false } ], "debug" : true, "params" : { "uri" :
"solr://collection1/select", "method" : "GET", "queryParams" : { "q" :
"a_txt:${doc_value}" }, "headers" : { } }, "skip" : false, "label" : "indexing-rpc"
}
```

Upload a stopword list:

```
{ "type" : "indexing-rpc", "id" : "demo", "debug" : true, "params" : { "uri" :
"http://FUSION_HOST:FUSION_PORT/api/stopwords/movies", "method" : "PUT", "headers"
: { "Content-Type" : "application/json" }, "entity" : "New stopword list" }, "skip"
: false, "label" : "Indexing RPC Demo", "type" : "indexing-rpc" }
```

Configuration

	When entering configuration values in the UI, use *unescaped* characters, such as `\t` for the tab character. When entering configuration values in the API, use *escaped* characters, such as `\\t` for the tab character.

Regex Field Extraction Index Stage

The Regex Field Extraction stage (called the Regular Expression Extractor stage in versions earlier than 3.0) is used to extract entities from documents based on matching regular expressions. The resulting regex matches over the contents of the source field are copied to the target field. The regular expression, source, and target fields are defined properties of this stage.

If using the REST API, this stage type is named "regex-extractor".

For examples of how to use this stage in the Fusion UI, see *Part 2* of the *Getting Started* tutorial.

Example Stage Specification

Define a regex-field-extraction stage to apply a regular expression that looks for storage capabilities of products when it appears in the product 'name' field, and store it in a special field:

```
{ "type" : "regex-field-extraction", "id" : "storagesize-regex-extraction", "rules"
: [ { "source" : [ "name" ], "target" : "storage_size_ss", "pattern" : "
(\\d{1,20}\\s{0,3}(GB|MB|TB|KB|mb|gb|tb|kb))", "annotateAs" : "storage_size" } ],
"skip" : false }
```

Configuration

When entering configuration values in the UI, use *unescaped* characters, such as `\t` for the tab character. When entering configuration values in the API, use *escaped* characters, such as `\\t` for the tab character.

Regex Field Filter Index Stage

The Regex Field Filter Index Stage (called the Regular Expression Filter stage in versions earlier than 3.0) removes a field or fields from a PipelineDocument according to a set of filters where each filter specifies a field name and a regular expression. If a field value matches the regular expression, the field is deleted from the document. The regex patterns follow Java regular expression pattern rules *(http://docs.oracle.com/javase/8/docs/api/java/util/regex/Pattern.html)*.

Example Stage Specification

Create a regex-filter to find Social Security Numbers and drop them from documents:

```
{ "type" : "regex-field-replacement", "id" : "ssnFilter", "skip" : false, "filters"
: [ { "sourceField" : "notes_t", "pattern" : "^\\d{3}-\\d{2}-\\d{4}$" } ] }
```

Configuration

When entering configuration values in the UI, use *unescaped* characters, such as `\t` for the tab character. When entering configuration values in the API, use *escaped* characters, such as `\\t` for the tab character.

Regex Field Replacement Index Stage

The Regex Field Replacement index stage lets you match against a source field and append or overwrite that field or another one with a replacement string. It uses the java.util.regex *(https://docs.oracle.com/javase/8/docs/api/java/util/regex/Pattern.html)* library.

One notable feature is support for flags in the prefix of the pattern, such as `(?iu)\w+` where the `i` flag means case-insensitive and `u` means Unicode-aware case folding. You can also configure the desired behavior when there is no match. Additionally, you can use capture groups in your replacement configuration using `$1` to denote the first capture group.

Configuration

> When entering configuration values in the UI, use *unescaped* characters, such as `\t` for the tab character. When entering configuration values in the API, use *escaped* characters, such as `\\t` for the tab character.

Resolve Multivalued Fields Index Stage

The Resolve Multivalued Fields stage (previously called the Multi-Value Resolver stage) allows you to choose one value from a set of one or more field values using a set of pre-defined rules, based on either field name or field type. For each field name or field type rule, a strategy is defined. There are 6 available strategies:

- DEFAULT: leave all values intact. This may cause problems if a field or dynamic field rule is not defined as multi-valued in the schema.
- PICK_FIRST: choose the first value and discard all others.
- PICK_LAST: choose the last value and discard all others.
- PICK_BY_CREATOR: based on the name of the "creator" metatdata, choose the name of 'creator' as defined in the creatorStrategy property for the field name. Only the last matching value will be retained.
- CONCATENATE_UNIQUE: choose to concatenate unique string values with a single space separator.
- PICK_MIN: choose the minimum value and discard all others. If the values are non-numeric, then all non-numeric values will be discarded.
- PICK_MAX: choose the maximum value and discard all others. If the values are non-numeric, then all non-numeric values will be discarded.

Configuration

> When entering configuration values in the UI, use *unescaped* characters, such as `\t` for the tab character. When entering configuration values in the API, use *escaped* characters, such as `\\t` for the tab character.

Send to Message Bus Index Stage

Send pipeline document to message bus.

Configuration

> When entering configuration values in the UI, use *unescaped* characters, such as `\t` for the tab character. When entering configuration values in the API, use *escaped* characters, such as `\\t` for the tab character.

Set Property Index Stage

The Set Property Index Stage is used to set a value on a document, or into the Pipeline Context for downstream consumption by specifying a series of simple matching conditions. These conditions match against whether a field exists, and simple substring matches on the field contents. For more complex logic, use a *JavaScript Index Stage*.

Configuration

> When entering configuration values in the UI, use *unescaped* characters, such as `\t` for the tab character. When entering configuration values in the API, use *escaped* characters, such as `\\t` for the tab character.

Solr Dynamic Field Name Mapping

This stage can be used to re-name existing fields to match Solr dynamic field names, so that they will be typed correctly by Solr. It also can be used to specify fields which should be exempt from name changes as well as to set the dynamic type suffixes to a specified field. Note that the field that is processed by a Field Mapping stage prior to this stage will not be renamed, even if the stage config matches the field name.

Configuration

	When entering configuration values in the UI, use *unescaped* characters, such as `\t` for the tab character. When entering configuration values in the API, use *escaped* characters, such as `\\t` for the tab character.

Solr Indexer Stage

A Solr Indexer stage transforms a Fusion PipelineDocument into a Solr document and sends it to Solr for indexing into a collection.

A PipelineDocument object contains fields which take as their values either a string or list of strings. Solr fields have a rich variety of types. The Solr Indexer stage transforms PipelineDocument field values into Solr document fields. The Solr Indexer stage can be configured so that it will try to ensure that all document fields are valid Solr fields.

This feature is convenient, but offers very little control over how fields and field values are transformed, especially with respect to dates. A *Date Parsing Index stage* offers greater control over date values. In a similar vein, Lucidworks recommends that you only use safe characters (a-z, _) in field names when using a Solr Indexer stage. Special characters will be automatically converted to an underscore.

Configuration

	When entering configuration values in the UI, use *unescaped* characters, such as `\t` for the tab character. When entering configuration values in the API, use *escaped* characters, such as `\\t` for the tab character.

Solr Partial Update Indexer Stage

The Solr Partial Update Indexer Stage updates of one or more fields of an existing Solr document in a collection managed by Fusion. It provides an alternative to the Solr Indexer stage.

When a data feed consists of an ongoing flow of messages about known documents in a collection, such as item price, inventory counts, or weather conditions at a location, this stage provides fast indexing throughput and can be configured to enforce data atomicity to guarantee that the index always reflects the most recent update.

This stage is configured with a set of update directives based on Solr's atomic updates *(https://cwiki.apache.org/confluence/display/solr/Updating+Parts+of+Documents)*. At run time, it creates a Solr update by applying these directive to the data from a Fusion PipelineDocument object and then submits this update to Solr's update handler.

	Solr's atomic update functionality requires that the schema for a collection is configured so that all fields have the attribute stored="true", excepting fields which are <copyField/> destinations which must be configured as stored="false".

Example Stage Specification

Configuration for a Partial Updater Stage in JSON:

```
{ "type" : "solr-partial-update-index", "enforceSchema" : false,
"solrDocIdFieldName" : "id", "solrDocIdFieldValue" : "<doc.id>", "updatedFields" :
[ { "updateType" : "set", "fieldName" : "statusValue", "values" : "
<doc.statusValue>" }, { "updateType" : "set", "fieldName" :
"lastCommunicationTime", "values" : "<doc.lastCommunicationTime>" } ],
"concurrencyControlEnabled" : true, "skip" : false, "label" : "solr-partial-update-
index", }
```

The expression <doc.X> will evaluate to the contents of the current PipelineDocument's field named "X".

Types of Update Operations

The set of update operations are based on operations supported by Solr. They are:

- 'add' - add a new value or values to an existing Solr document field, or add a new field and value(s).
- 'set' - change the value or values in an existing Solr document field.
- 'remove' - remove all occurrences of the value or values from an existing Solr document field.
- 'removeregex' - remove all occurrences of the values which match the regex or list of regexes from an existing Solr document field.
- 'increment' - increment the the numeric value of existing Solr document field by a specific amount.
- 'decrement' - decrement the the numeric value of existing Solr document field by a specific amount.

In addition, this stage introduces experimental "Positional" operations which can be used to add, set or remove exactly one element of a field which takes a list of values (i.e, a multi-valued field).

- 'positionalUpdates' - used to add or set the value at specific position.
- 'positionalRemoves' - used to delete an element at a specific position.

When a collection contains two or more multi-value fields which are maintained in parallel so that taken together, they act like a table stored column by column, a positional update operation updates several data cells across one row of the table. To maintain this kind of column-oriented table, the positional delete directive must specify all the fields in the document which logically comprise the table.

Document Identifier Field

A Fusion collection is a Solr collection managed by Fusion. Underlyingly, a Solr document is a list of named, typed fields. The Solr unique key field *(https://wiki.apache.org/solr/UniqueKey)* stores a string which is the unique identifier for that document. There is at most one UniqueKey field per document, which is defined in the Solr schema. The UniqueKey field value is required. For collections created via Fusion, the UniqueKey field is named "id". Other document fields may also store string values which can be used as a unique identifier.

Solr uses the UniqueKey field to find the document to be updated. If the data feed information contains a document identifier which is different than the identifier value stored in the UniqueKey field, then this stage must do a Solr lookup to find the UniqueKey value.

Optimistic Concurrency

Solr's Optimistic Concurrency *(http://yonik.com/solr/optimistic-concurrency/)* is a mechanism which checks whether or not a document has changed between the point at which an update request was submitted and the point at which the request is processed. Solr documents have an internal field named "_version_" which is updated whenever there is any change made to any of the other fields in that document. When optimistic concurrency control is on, update requests will be discarded if the current version of the document has changed since that request was made. This guarantees that the document will always reflect the most recent update. However, this require an additional Solr lookup to get the current document version number, which is submitted as part of the update request.

Performance Considerations

In order to send a single update request to Solr, without preliminary lookup requests:

- The document identifier field should match the Solr collection's UniqueKey identifier field.
- Optimistic Concurrency should be turned off.
- Positional updates are experimental and potentially expensive, since all the values for all fields being updated must be fetched into memory in order to perform positional operations.

Solr Date Formats

```
"yyyy-MM-dd'T'HH:mm:ss'Z'", // Solr format without milliseconds "yyyy-MM-
dd'T'HH:mm:ss.SSS'Z'", // standard Solr format, with literal "Z" at the end "yyyy-
MM-dd'T'HH:mm:ss.SS'Z'", // standard Solr format, with literal "Z" at the end
"yyyy-MM-dd'T'HH:mm:ss.S'Z'" // standard Solr format, with literal "Z" at the end
```

See https://cwiki.apache.org/confluence/display/solr/Working+with+Dates
(https://cwiki.apache.org/confluence/display/solr/Working+with+Dates)

Configuration

When entering configuration values in the UI, use *unescaped* characters, such as `\t` for the tab character. When entering configuration values in the API, use *escaped* characters, such as `\\t` for the tab character.

Text Processing Index Stage

The Text Processing Index Stage processes simple text according to the transformation applied in a field.

This stage requires the Tika service in order to be deployed inside the cluster. It will only be deployed inside the cluster *by default* in environments where **Transport Layer Security** has been enabled*.

Update Experiment Stage

Configuration

When entering configuration values in the UI, use *unescaped* characters, such as `\t` for the tab character. When entering configuration values in the API, use *escaped* characters, such as `\\t` for the tab character.

Update Related Document Stage

This index pipeline stage performs a Solr atomic update on an existing document using data pulled from the current document being processed by the pipeline.

Configuration

When entering configuration values in the UI, use *unescaped* characters, such as `\t` for the tab character. When entering configuration values in the API, use *escaped* characters, such as `\\t` for the tab character.

XML Transformation Index Stage

The XML Transformation stage (previously called the XML Transform Stage) allows you to process an XML document into one or more Solr documents and to specify mappings between elements and document fields. A common use case for an XML Transformation stage in a pipeline is when the XML document is a container-like document which contains a set of inner elements, each of which should be treated as a separate document. A parent ID field can be used to relate these multiple documents back to the containing document.

Pipeline Configuration

The default XML processing provided by the Apache Tika Parser index stage extracts all text from an XML into a single document field called `content`. This not only flattens the document contents, it loses all information about the containing elements in the document. To process XML documents using an XML Transformation stage, the index pipeline must have as its initial processing stage an Apache Tika Parser index stage which is configured to pass the document through to the XML Transformation stage *as raw XML*, via the following configuration:

- UI checkbox "Add original document content" **unchecked** / REST API property "addOriginalContent" set to **false**
- UI checkbox "Return parsed content as XML or HTML" **checked** / REST API property "keepOriginalStructure" set to **true**
- UI checkbox "Return original XML and HTML instead of Tika XML output" **checked** / REST API property "returnXml" set to **true**

With this configuration, the Tika parser stage decodes the raw input stream of bytes into a string containing the entire XML document which is returned in the PipelineDocument field `body`.

The pipeline must have a Field Mapping stage after the XML Transformation stage, before the Solr Indexer stage. The Field Mapping stage is used to remove the following fields from the document:

- *raw-content*
- Content-Type
- Content-Length
- parsing
- parsing_time

XML Transforms

The XML Transformation stage uses a Solr XPathRecordReader *(http://lucene.apache.org/solr/6_1_0/solr-dataimporthandler/org/apache/solr/handler/dataimport/XPathRecordReader.html)* which is a streaming XML parser that supports *only a limited subset of XPath selectors*. It provides exact matching on element attributes and it can only extract the element text, not attribute values.

Examples of allowed XPath specifications where "a", "b", "c" are any element tags, likewise "attrName" is any attribute name:

```
/a/b/c /a/b/c[@attrName='someValue'] /a/b/c[@attrName=]/d /a/b/c/@attrName
//b//...
```

	When specifying the list of `mappings`, for each mapping, the specification for the `xpath` attribute must include the full path, i.e., the `xpath` attribute will include the `rootXPath`. See the example configuration below.

Example Stage Specification

Definition of an XML-Transformation stage that extracts elements from a MEDLINE/Pubmed article abstract:

```
{ "type" : "xml-transform", "id" : "n0j2a9k9", "rootXPath" :
"/MedlineCitationSet/MedlineCitation", "bodyField" : "body", "mappings" : [ {
"xpath" : "/MedlineCitationSet/MedlineCitation/Article/ArticleTitle", "field" :
"article-title_txt", "multivalue" : false }, { "xpath" :
"/MedlineCitationSet/MedlineCitation/Article/Abstract/AbstractText", "field" :
"article-abstract_txt", "multivalue" : true }, { "xpath" :
"/MedlineCitationSet/MedlineCitation/MeshHeadingList/MeshHeading/DescriptorName",
"field" : "mesh-heading_txt", "multivalue" : true }, { "xpath" :
"/MedlineCitationSet/MedlineCitation/PMID", "field" : "pmid_txt", "multivalue" :
false } ], "keepParent" : false, "skip" : false, "label" : "medline_xml_transform",
}
```

Template for a minimal index pipeline that includes an XML-Transformation stage. Replace the XPath and field names in the XML-Transformation stage according to your data.

```
{ "id" : "xml-pipeline-default", "stages" : [ { "type" : "tika-parser",
"includeImages" : false, "flattenCompound" : false, "addFailedDocs" : false,
"addOriginalContent" : false, "contentField" : "_raw_content_", "returnXml" : true,
"keepOriginalStructure" : true, "extractHtmlLinks" : false, "extractOtherLinks" :
false, "csvParsing" : false, "skip" : false, "label" : "tika", "sourceField" :
"_raw_content_" }, { "type" : "xml-transformation", "rootXPath" : "/ROOTS/ROOT",
"bodyField" : "body", "mappings" : [ { "xpath" : "/ROOTS/ROOT/element", "field" :
"element-field_t", "multivalue" : false } ], "keepParent" : false, "skip" : false,
"label" : "xml" }, { "type" : "field-mapping", "mappings" : [ { "source" :
"parsing", "operation" : "delete" }, { "source" : "parsing_time", "operation" :
"delete" }, { "source" : "Content-Type", "operation" : "delete" }, { "source" :
"Content-Length", "operation" : "delete" } ], "skip" : false, "label" : "field
mapping" }, { "type" : "solr-index", "enforceSchema" : true, "bufferDocsForSolr" :
false, "skip" : false, "label" : "solr-index" } ] }
```

Configuration

	When entering configuration values in the UI, use *unescaped* characters, such as `\t` for the tab character. When entering configuration values in the API, use *escaped* characters, such as `\\t` for the tab character.

Custom JavaScript Stages Global Variables

JavaScript Index Stage Global Variables

JavaScript *(http://en.wikipedia.org/wiki/JavaScript)* is a lightweight scripting language. The JavaScript in a JavaScript stage is standard ECMAScript *(http://en.wikipedia.org/wiki/ECMAScript)*. What a JavaScript program can do depends on the container in which it runs. For a JavaScript Index stage, the container is a Fusion index pipeline. The following global pipeline variables are available:

Name	Type	Description
`doc`	*Pipeline Document*	The contents of each document submitted to the pipeline.
`ctx`	*Context*	A map that stores miscellaneous data created by each stage of the pipeline. Use the `ctx` variable instead of the deprecated `_context` global variable. The `ctx` variable is used to: • Pass data from one stage to another • Store data that needs to be passed from one custom stage to a later custom stage The data can differ between stages: • If the previous stage changes the data • Based on the configuration of each stage If the data is modified in one stage, it may cause a later stage to function irregularly.
`collection`	String	The name of the Fusion collection being indexed or queried.
`solrServer`	*Buffering solrServer*	The Solr server instance that manages the pipeline's default Fusion collection. All indexing and query requests are done by calls to methods on this object. See solrClient *(https://lucene.apache.org/solr/5_2_1/solr-solrj/org/apache/solr/client/solrj/SolrClient.html)* for details
`solrServerFactory`	*solrClusterComponent*	The SolrCluster server used for lookups by collection name which returns a Solr server instance for that collection. For example: `var productsSolr = solrServerFactory.getSolrServer("products");`.

Syntax Variants

JavaScript stages can be written using legacy syntax or function syntax. The key difference between these syntax variants is how the "global variables" are used. While using legacy syntax, these variables are used as global variables. With function syntax, however, these variables are passed as function parameters.

Legacy Syntax

Legacy syntax is used to perform very simple tasks.

```
//do some work... return doc;
```

Function Syntax

Function syntax is used for moderately complex tasks.

```
function (doc) { // do some work ... return doc; }
```

Function syntax is used for the examples in this document.

Advanced Syntax

Advanced syntax is used for complex tasks and when multiple functions are needed.

```
/* globals Java, logger*/ (function () { "use strict"; return function main(doc ,
ctx, collection, solrServer, solrServerFactory) { // do some work ... return doc;
}; })();
```

JavaScript Use

The JavaScript in a JavaScript Index stage must return either a single document or an array of documents. This can be accomplished by either:

- a series of statements where the final statement evaluates to a document or array of documents
- a function that returns a document or an array of documents

All pipeline variables referenced in the body of the JavaScript function are passed in as arguments to the function. For example, in order to access the PipelineDocument in global variable 'doc', the JavaScript function is written as follows:

```
function (doc) { // do some work ... return doc; }
```

The allowed set of function declarations are:

```
function doWork(doc) { ... return doc; } function doWork(doc, ctx) { ... return
doc; } function doWork(doc, ctx, collection) { ... return doc; } function
doWork(doc, ctx, collection, solrServer) { ... return doc; } function doWork(doc,
ctx, collection, solrServer, SolrClusterComponent) { ... return doc; }
```

The order of these arguments is according to the (estimated) frequency of use. The assumption is that most processing only requires access to the document object itself, and the next-most frequent type of processing requires only the document and read-only access of some context parameters. If you need to reference the solrServerFactory global variable, you must use the 5-arg function declaration.

In order to use other functions in your JavaScript program, you can define and use them, as long as the final statement in the program returns a document or documents.

Global variable `logger`

The logs are output to the indexing service logs for custom index stages. Access the Log Viewer and filter on this service to view the information.

JavaScript Query Stage Global Variables

JavaScript *(http://en.wikipedia.org/wiki/JavaScript)* is a lightweight scripting language. The JavaScript in a JavaScript stage is standard ECMAScript *(http://en.wikipedia.org/wiki/ECMAScript)*. What a JavaScript program can do depends on the container in which it runs. For

a JavaScript Query stage, the container is a Fusion query pipeline. The following global pipeline variables are available:

Name	Type	Description
`request`	*Request for a Regular Query*	The `request` variable contains Solr query information and is referred to as a **regular** request. A regular query request does *not* include Search DSL (domain specific language) parameters.
`request`	*DSL Request*	The `request` variable is also used when the query contains parameters for a Search DSL (domain specific language) request. See *Domain Specific Language* for more information.
`response`	*Response for a Regular Query*	The `response` variable contains Solr response information, and is used to return information from a **regular** request. Because a regular request does *not* include Search DSL (domain specific language) parameters, the response will not return Search DSL results.
`response`	*DSL Response*	The `response` variable is also used to return information from a Search DSL (domain specific language) request. See *Domain Specific Language* for more information.
`ctx`	*Context*	A map that stores miscellaneous data created by each stage of the pipeline. Use the `ctx` variable instead of the deprecated `_context` global variable. The `ctx` variable is used to: • Pass data from one stage to another • Store data that needs to be passed from one custom stage to a later custom stage The data can differ between stages: • If the previous stage changes the data • Based on the configuration of each stage If the data is modified in one stage, it may cause a later stage to function irregularly.
`collection`	String	The name of the Fusion collection being indexed or queried.
`solrServer`	*BufferingsolrServer*	The Solr server instance that manages the pipeline's default Fusion collection. All indexing and query requests are done by calls to methods on this object. See solrClient *(https://lucene.apache.org/solr/5_2_1/solr-solrj/org/apache/solr/client/solrj/SolrClient.html)* for details.
`solrServerFactory`	*solrClusterComponent*	The SolrCluster server used for lookups by collection name which returns a Solr server instance for that collection, e.g. `var productsSolr = solrServerFactory.getSolrServer("products");`.
`QueryRequestAndResponse`	*QueryRequestAndResponse*	Used to create query pipeline requests and responses.
`JSONResponse`	*JSONResponse*	Returns list of info as a string.

See *Custom JavaScript Query Stage Examples* for more information.

Syntax Variants

JavaScript stages can be written using legacy syntax or function syntax. The key difference between these syntax variants is how the "global variables" are used. While using legacy syntax, these variables are used as global variables. With function syntax, however, these variables are passed as function parameters.

Legacy Syntax

Legacy syntax is used to perform very simple tasks.

```
//do some work... return doc;
```

Function Syntax

```
function(request,response) { request.addParam("foo", "bar"); }
```

Function syntax is used for the examples in this document.

Global variable `logger`

The logs are output to the query service logs for custom query stages. Access the Log Viewer and filter on this service to view the information.

Parsers

Parsers provide fine-grained configuration for inbound data. You configure parsers with stages, much like index pipelines and query pipelines. Parsers can include conditional parsing and nested parsing. You can configure them through the *Fusion UI* or the *Parsers API*.

Connectors receive the inbound data, convert it into a byte stream, and send the byte stream to a parser's configured parsing stages. The parser selects a parsing stage to handle the stream, which parses the data and produces documents that are sent to the index pipeline.

Each parsing stage evaluates whether the inbound stream matches the stage's default media types or filename extensions. The first stage that finds a match processes the data and can output one or both of the following:

- Zero or more pipeline documents for consumption by the index pipeline
- Zero or more new input streams for re-parsing
  This recursive approach is useful for containers (for example, `zip` and `tar` files). The output of the container parsing can be another container or a stream of uncompressed content that requires its own parsing.

Stages that might match the stream beyond the first match will not be used.

A few static fields impact the overall parser configuration. They are accessible when you select the parser in the Index Workbench:

Field	Description
Document ID Source Field	Field in the source file that contains the document ID
Maximum Parser Recursion Depth	Maximum number of times the parser may recurse over the file, before proceeding to the next parser. This is useful for files with hierarchical structures (for example, `zip` and `tar` files).
Enable automatic media type detection	Whether to automatically detect the media type of the source files. If disabled, the parser uses the media type `application/octet-stream`.

Built-in parser stages

The parser stages found in the sidenav are available for configuration.

Datasources that use connectors that retrieve fixed-structure content (such as those for Twitter and Jira) have hard-coded parsers and do not expose any configurable parser options.

Configure parsers

When you configure a datasource, you can use the *Index Workbench* or the *Parsers API* to create a parser. A parser consists of an ordered list of parser stages, some global parser parameters, and the stage-specific parameters. You can re-order the stages list by dragging them up or down in the Index Workbench.

Any parser stage can be added to the same parser multiple times if different configuration options are needed for different stages. Datasources with fixed-structure data will also be parsed by Fusion, but with default settings that do not need to be customized.

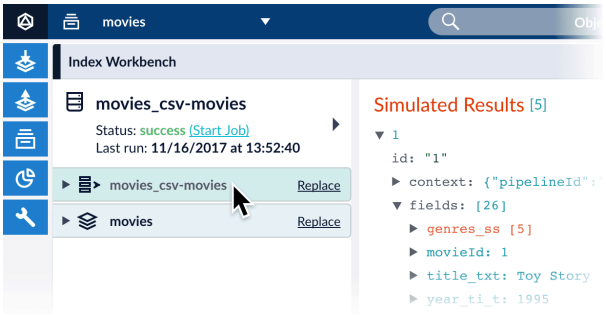
There is no limit to the number of stages that can be included in a parser. The priority-order of the stages is completely flexible. In a default parser configuration, a fallback parser is provided at the end of the parsing stage list to handle streams no other stage matches. If present, this stage is selected and attempts to parse anything that has not yet been matched.

> When entering configuration values in the UI, use *unescaped* characters, such as `\t` for the tab character. When entering configuration values in the API, use *escaped* characters, such as `\\t` for the tab character.

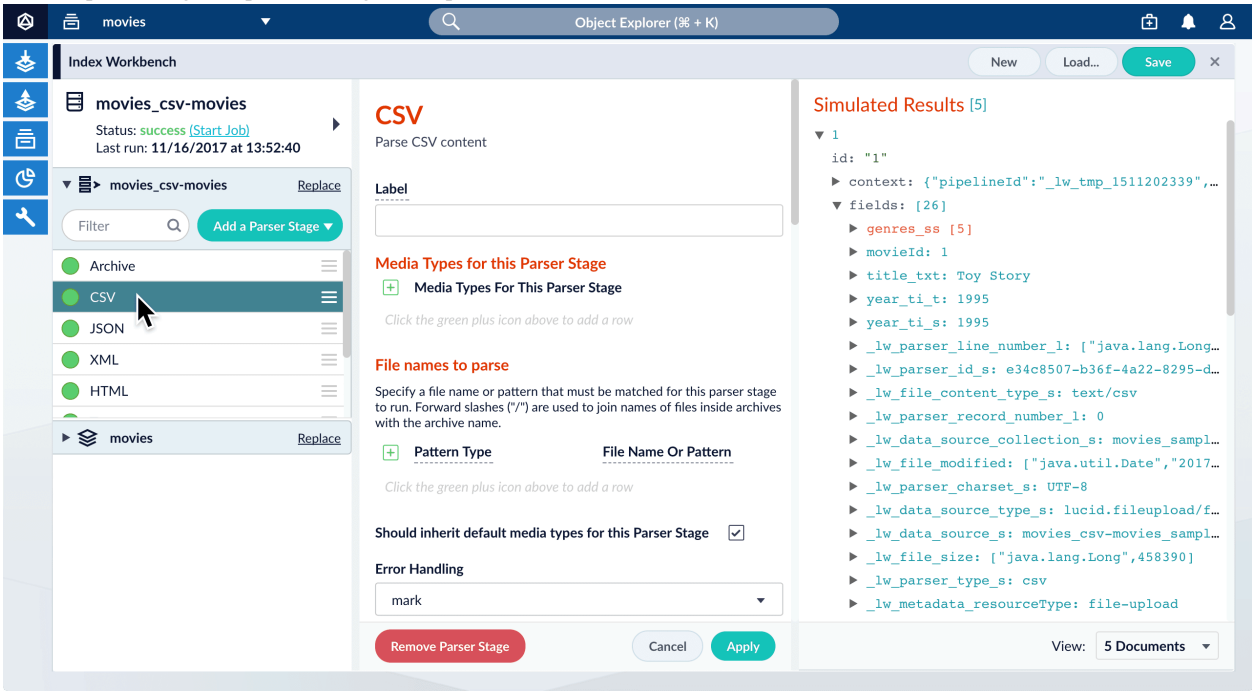
Configure a parser in the Fusion UI

To configure parser stages using the Fusion UI:

1. In the *Fusion workspace*, navigate to the *Index Workbench*.
2. At the upper right of the Index Workbench panel, click **Load**.
3. Under **Load**, click the name of the index pipeline.
4. Click the parser to open its configuration:



5. Click a specific stage to open its configuration panel:



Configure a parser in the REST API

The *Parsers API* provides a programmatic interface for viewing, creating, and modifying parsers, as well as sending documents directly to a parser.

- To get all currently-defined parsers: `http://FUSION_HOST:FUSION_PORT/api/parsers/`
- To get the parser schema: `http://FUSION_HOST:FUSION_PORT/api/parsers/_schema`

Here is a very simple parser example, for parsing JSON input:

```
{ "id": "simple-json", "type": "json", arbitrary parser-specific options here.
"prettify": false }
```

The example below shows a parser that can parse JSON input, as well as JSON that is inside zip, tar, or gzip containers, or any combination (such as .tar.gz). The order of the stages begins with the outermost containers and ends with the innermost content.

```
{ "id": "default-json", "type": "composite", "parsers": [ { "id" : "zip-parser",
"type" : "zip" }, { "type" : "gz" }, { "type" : "tar" }, { "id": "json-parser",
"type": "json", "prettify": false }] }
```

ID is optional, just as in pipeline stages. Many parser stages require no configuration other than `type`.

Field parser index pipeline stage

The parsers themselves only parse whole documents. Parsing of content embedded in fields is performed separately by the Field Parser index pipeline stage. This stage identifies the field that requires parsing and the parser to use.

Apache Tika Parser

Apache Tika is a versatile parser that supports many types of unstructured document formats, such as HTML, PDF, Microsoft Office, OpenOffice, RTF, audio, video, images, and more. A complete list of supported formats is available at Apache Tika *(http://tika.apache.org/)*.

To perform image text extraction when **Include images** is enabled, install Tesseract *(https://github.com/tesseract-ocr/tesseract)* in the server hosting Fusion.

This stage is not compatible with asynchronous Tika parsing.

When entering configuration values in the UI, use *unescaped* characters, such as `\t` for the tab character. When entering configuration values in the API, use *escaped* characters, such as `\\t` for the tab character.

Apache Tika Server Stage

Apache Tika Server is a versatile parser that supports parsing many document formats.

This stage is not compatible with asynchronous Tika parsing.

When entering configuration values in the UI, use *unescaped* characters, such as `\t` for the tab character. When entering configuration values in the API, use *escaped* characters, such as `\\t` for the tab character.

Archive Parser Stage

The Archive parser stage can parse the majority of common archive and compressed file formats. They are parsed into their constituent documents, which can then be parsed further or sent straight to the index pipeline. The following archive formats are supported:

- tar
- zip
- jar
- 7z
- ar
- arj
- Unix dump
- cpio

When entering configuration values in the UI, use *unescaped* characters, such as `\t` for the tab character. When entering configuration values in the API, use *escaped* characters, such as `\\t` for the tab character.

CSV Parser Stage

This parser breaks down incoming CSV files into the most efficient components for Fusion to index. It produces one new document per row from the CSV input, excluding comment rows and header rows.

When entering configuration values in the UI, use *unescaped* characters, such as `\t` for the tab character. When entering configuration values in the API, use *escaped* characters, such as `\\t` for the tab character.

Fallback Parser Stage

The Fallback parser stage is useful for processing data that Fusion does not have a specified parsing process for. Fallback does not technically parse data, since it does not know what to do with it, it simply copies the raw bytes into a Solr document. If your Fusion parser stage configuration encounters data it does not know how to parse, such as someone's proprietary data file format, it will copy it as-is, whereas if it encounters recognizable data in more common file types, such as PDFs, Fusion will parse the text and metadata using Tika.

The Fallback parser acts as the final stage that attempts to parse any documents that haven't been parsed already. More commonly, the parser selects a stage specifically built to handle the data and runs that stage.

When entering configuration values in the UI, use *unescaped* characters, such as `\t` for the tab character. When entering configuration values in the API, use *escaped* characters, such as `\\t` for the tab character.

Forked Apache Tika Parser Stage

A simplified Apache Tika parser more geared towards an Enterprise Search crawl where all documents are parsed in a forked Java Virtual Machine. If memory issues occur during the parse, such as an Out Of Memory condition, the connector job is unaffected.

	When entering configuration values in the UI, use *unescaped* characters, such as `\t` for the tab character. When entering configuration values in the API, use *escaped* characters, such as `\\t` for the tab character.

Grok Parser Stage

The Grok parser stage uses Java Grok *(https://github.com/thekrakken/java-grok)* and Grok patterns (a specific kind of regex matching) to parse log files and similar text files that have line-oriented, semi-structured data. Parsing a text file with the Grok parser lets you give more structure to semi-structured data and extract more information.

Whether the Grok stage parses a file

Before a Grok parser stage parses a file, the file must meet criteria regarding the media type and file name.

Media type

The Grok parser stage parses files that have media types that match either the default media types *or* media types that you specify.

Select or unselect **Use default media types for this parser stage**:

- Selected.* The Grok parser stage parses files that have one of the default media types (`text/plain` or `text/x-log`), as well as files that have media types that you specify under **Media Types for this Parser Stage**.
- Unselected.* The Grok parser stage *only* parses files that have one of the media types that you specify under **Media Types for this Parser Stage**.

File name

Optionally, you can specify a file name or file name pattern that a file must match for the Grok parser stage to parse the file.

Field	Description
Pattern Type	`glob`. Use `bash` shell wildcards. Examples include `z.txt`, `*.md`, and `/a/*/b/f.txt`. `regex`. Use Java regular expressions (PCRE; Perl-compatible regular expressions). Examples include `z.txt$`, `.*\.txt$`, and `^/a/[^\/]*/b/f.txt$`.
File Name or Pattern	Name of the file or a pattern for the file name. The parser parses matching files.

Grok patterns

Grok patterns are regular expressions written in the language of the Oniguruma regular expression library *(https://github.com/kkos/oniguruma)*, which has this syntax *(https://github.com/kkos/oniguruma/blob/master/doc/RE)*.

You configure a Grok parsing stage to use predefined *Grok patterns* (about 300 patterns are available) and/or Grok pattern definitions that you write yourself.

- **Use predefined patterns.** Under the **Grok Pattern** part of the Grok parser stage configuration, specify a single top-level Grok pattern by name, for example, `REDISLOG`.
- **Write your own Grok pattern definition(s).** (*optional*) Write one or more Grok pattern definitions, and then enter them in the **Grok Definition** part of the Grok parser stage configuration.

Parsing rules

These are rules that affect the results of parsing:

- **Precedence in the event of identical names.** If the name of a custom Grok pattern definition that you provide is identical to the name of a predefined pattern definition, then your definition is used.
- **Invalid patterns.** If a pattern is not syntactically valid, then the full text of the row being parsed is treated as a single field.
- **Pattern does not match any data.** If a pattern does not match any data, then the full text of the row being parsed is treated as a single field.
- **Line by line.** Parsing is line by line. If data has a multiline structure, the parser does not capture the relationship between lines.

	When entering configuration values in the UI, use *unescaped* characters, such as `\t` for the tab character. When entering configuration values in the API, use *escaped* characters, such as `\\t` for the tab character.

Grok Patterns

Below we list predefined Grok patterns that you can use with the Grok parser. The patterns are grouped by the kinds of files in which they occur.

Solr

```
HANDLER %{WORD}.%{WORD}.%{WORD}.%{WORD}.%{WORD} SOLR_QUERY \[collection%
{NUMBER:collection}\]\s+webapp=/%{WORD:webapp}\s+path=/%{WORD:path}\s+params=\{%
{GREEDYDATA:params}\}\s+hits=%{NUMBER:hits}\s+status=%{NUMBER:status}\s+QTime=%
{NUMBER:QTime} SOLR_LOG %{LOGLEVEL}\s+-\s%{TIMESTAMP_ISO8601};\s%{JAVACLASS};\s+%
{MESSAGE:message} SOLR_6_LOG4J %{TIMESTAMP_ISO8601:logdate} %{LOGLEVEL:level_s} \(%
{DATA:thread_s}\) \[(?:%{DATA:mdc_s}| )\] %{DATA:category_s} %
{JAVALOGMESSAGE:logmessage} SOLR_6_REQUEST \[%{DATA:core_node_name_s}\] webapp=\/%
{WORD:webapp_s} path=\/%{DATA:path_s} params=\{%{GREEDYDATA:params_s}\} hits=%
{NUMBER:hits_i} status=%{NUMBER:status_i} QTime=%{NUMBER:qtime_i}
```

Ruby

```
RUBY_LOGLEVEL (?:DEBUG|FATAL|ERROR|WARN|INFO) RUBY_LOGGER [DFEWI], \[%
{TIMESTAMP_ISO8601:timestamp} #%{POSINT:pid}\] *%{RUBY_LOGLEVEL:loglevel} -- +%
{DATA:progname}: %{GREEDYDATA:message} REDISTIMESTAMP %{MONTHDAY} %{MONTH} %{TIME}
REDISLOG \[%{POSINT:pid}\] %{REDISTIMESTAMP:timestamp} \*
```

Rails

```
RUUID \h{32} # rails controller with action RCONTROLLER (?<controller>[^#]+)#(?
<action>\w+) # this will often be the only line: RAILS3HEAD (?m)Started %
{WORD:verb} "%{URIPATHPARAM:request}" for %{IPORHOST:clientip} at (?<timestamp>%
{YEAR}-%{MONTHNUM}-%{MONTHDAY} %{HOUR}:%{MINUTE}:%{SECOND} %{ISO8601_TIMEZONE}) #
for some a strange reason, params are stripped of {} - not sure that is a good
idea. RPROCESSING \W*Processing by %{RCONTROLLER} as (?<format>\S+)
(?:\W*Parameters: {%{DATA:params}}\W*)? RAILS3FOOT Completed %{NUMBER:response}%
{DATA} in %{NUMBER:totalms}ms %{RAILS3PROFILE}%{GREEDYDATA} RAILS3PROFILE (?:\
(Views: %{NUMBER:viewms}ms \| ActiveRecord: %{NUMBER:activerecordms}ms|\
(ActiveRecord: %{NUMBER:activerecordms}ms)? # putting it all together RAILS3 %
{RAILS3HEAD}(?:%{RPROCESSING})?(?<context>(?:%{DATA}\n)*)(?:%{RAILS3FOOT})?
```

Postgressql

```
# Default postgresql pg_log format pattern POSTGRESQL %{DATESTAMP:timestamp} %{TZ}
%{DATA:user_id} %{GREEDYDATA:connection_id} %{POSINT:pid}
```

Nagios

```
############################################################################################
############################################################################################
# Chop Nagios log files to smithereens! # # A set of GROK filters to process
logfiles generated by Nagios. # While it does not, this set intends to cover all
possible Nagios logs. # # Some more work needs to be done to cover all External
Commands: # http://old.nagios.org/developerinfo/externalcommands/commandlist.php #
# If you need some support on these rules please contact: # Jelle Smet
http://smetj.net #
############################################################################################
############################################################################################
NAGIOSTIME \[%{NUMBER:nagios_epoch}\]
############################################### ######## Begin nagios log types
############################################### NAGIOS_TYPE_CURRENT_SERVICE_STATE
CURRENT SERVICE STATE NAGIOS_TYPE_CURRENT_HOST_STATE CURRENT HOST STATE
NAGIOS_TYPE_SERVICE_NOTIFICATION SERVICE NOTIFICATION NAGIOS_TYPE_HOST_NOTIFICATION
HOST NOTIFICATION NAGIOS_TYPE_SERVICE_ALERT SERVICE ALERT NAGIOS_TYPE_HOST_ALERT
HOST ALERT NAGIOS_TYPE_SERVICE_FLAPPING_ALERT SERVICE FLAPPING ALERT
NAGIOS_TYPE_HOST_FLAPPING_ALERT HOST FLAPPING ALERT
NAGIOS_TYPE_SERVICE_DOWNTIME_ALERT SERVICE DOWNTIME ALERT
NAGIOS_TYPE_HOST_DOWNTIME_ALERT HOST DOWNTIME ALERT
NAGIOS_TYPE_PASSIVE_SERVICE_CHECK PASSIVE SERVICE CHECK
NAGIOS_TYPE_PASSIVE_HOST_CHECK PASSIVE HOST CHECK NAGIOS_TYPE_SERVICE_EVENT_HANDLER
SERVICE EVENT HANDLER NAGIOS_TYPE_HOST_EVENT_HANDLER HOST EVENT HANDLER
NAGIOS_TYPE_EXTERNAL_COMMAND EXTERNAL COMMAND NAGIOS_TYPE_TIMEPERIOD_TRANSITION
TIMEPERIOD TRANSITION ############################################### ######## End
nagios log types ###############################################
############################################### ######## Begin external check types
############################################### NAGIOS_EC_DISABLE_SVC_CHECK
DISABLE_SVC_CHECK NAGIOS_EC_ENABLE_SVC_CHECK ENABLE_SVC_CHECK
NAGIOS_EC_DISABLE_HOST_CHECK DISABLE_HOST_CHECK NAGIOS_EC_ENABLE_HOST_CHECK
ENABLE_HOST_CHECK NAGIOS_EC_PROCESS_SERVICE_CHECK_RESULT
PROCESS_SERVICE_CHECK_RESULT NAGIOS_EC_PROCESS_HOST_CHECK_RESULT
PROCESS_HOST_CHECK_RESULT NAGIOS_EC_SCHEDULE_SERVICE_DOWNTIME
SCHEDULE_SERVICE_DOWNTIME NAGIOS_EC_SCHEDULE_HOST_DOWNTIME SCHEDULE_HOST_DOWNTIME
NAGIOS_EC_DISABLE_HOST_SVC_NOTIFICATIONS DISABLE_HOST_SVC_NOTIFICATIONS
NAGIOS_EC_ENABLE_HOST_SVC_NOTIFICATIONS ENABLE_HOST_SVC_NOTIFICATIONS
NAGIOS_EC_DISABLE_HOST_NOTIFICATIONS DISABLE_HOST_NOTIFICATIONS
NAGIOS_EC_ENABLE_HOST_NOTIFICATIONS ENABLE_HOST_NOTIFICATIONS
NAGIOS_EC_DISABLE_SVC_NOTIFICATIONS DISABLE_SVC_NOTIFICATIONS
NAGIOS_EC_ENABLE_SVC_NOTIFICATIONS ENABLE_SVC_NOTIFICATIONS
############################################### ######## End external check types
############################################### NAGIOS_WARNING Warning:%{SPACE}%
{GREEDYDATA:nagios_message} NAGIOS_CURRENT_SERVICE_STATE %
{NAGIOS_TYPE_CURRENT_SERVICE_STATE:nagios_type}: %{DATA:nagios_hostname};%
{DATA:nagios_service};%{DATA:nagios_state};%{DATA:nagios_statetype};%
{DATA:nagios_statecode};%{GREEDYDATA:nagios_message} NAGIOS_CURRENT_HOST_STATE %
{NAGIOS_TYPE_CURRENT_HOST_STATE:nagios_type}: %{DATA:nagios_hostname};%
{DATA:nagios_state};%{DATA:nagios_statetype};%{DATA:nagios_statecode};%
{GREEDYDATA:nagios_message} NAGIOS_SERVICE_NOTIFICATION %
{NAGIOS_TYPE_SERVICE_NOTIFICATION:nagios_type}: %{DATA:nagios_notifyname};%
{DATA:nagios_hostname};%{DATA:nagios_service};%{DATA:nagios_state};%
{DATA:nagios_contact};%{GREEDYDATA:nagios_message} NAGIOS_HOST_NOTIFICATION %
{NAGIOS_TYPE_HOST_NOTIFICATION:nagios_type}: %{DATA:nagios_notifyname};%
{DATA:nagios_hostname};%{DATA:nagios_state};%{DATA:nagios_contact};%
{GREEDYDATA:nagios_message} NAGIOS_SERVICE_ALERT %
{NAGIOS_TYPE_SERVICE_ALERT:nagios_type}: %{DATA:nagios_hostname};%
{DATA:nagios_service};%{DATA:nagios_state};%{DATA:nagios_statelevel};%
{NUMBER:nagios_attempt};%{GREEDYDATA:nagios_message} NAGIOS_HOST_ALERT %
{NAGIOS_TYPE_HOST_ALERT:nagios_type}: %{DATA:nagios_hostname};%
{DATA:nagios_state};%{DATA:nagios_statelevel};%{NUMBER:nagios_attempt};%
{GREEDYDATA:nagios_message} NAGIOS_SERVICE_FLAPPING_ALERT %
{NAGIOS_TYPE_SERVICE_FLAPPING_ALERT:nagios_type}: %{DATA:nagios_hostname};%
{DATA:nagios_service};%{DATA:nagios_state};%{GREEDYDATA:nagios_message}
```

```
NAGIOS_HOST_FLAPPING_ALERT %{NAGIOS_TYPE_HOST_FLAPPING_ALERT:nagios_type}: %
{DATA:nagios_hostname};%{DATA:nagios_state};%{GREEDYDATA:nagios_message}
NAGIOS_SERVICE_DOWNTIME_ALERT %{NAGIOS_TYPE_SERVICE_DOWNTIME_ALERT:nagios_type}: %
{DATA:nagios_hostname};%{DATA:nagios_service};%{DATA:nagios_state};%
{GREEDYDATA:nagios_comment} NAGIOS_HOST_DOWNTIME_ALERT %
{NAGIOS_TYPE_HOST_DOWNTIME_ALERT:nagios_type}: %{DATA:nagios_hostname};%
{DATA:nagios_state};%{GREEDYDATA:nagios_comment} NAGIOS_PASSIVE_SERVICE_CHECK %
{NAGIOS_TYPE_PASSIVE_SERVICE_CHECK:nagios_type}: %{DATA:nagios_hostname};%
{DATA:nagios_service};%{DATA:nagios_state};%{GREEDYDATA:nagios_comment}
NAGIOS_PASSIVE_HOST_CHECK %{NAGIOS_TYPE_PASSIVE_HOST_CHECK:nagios_type}: %
{DATA:nagios_hostname};%{DATA:nagios_state};%{GREEDYDATA:nagios_comment}
NAGIOS_SERVICE_EVENT_HANDLER %{NAGIOS_TYPE_SERVICE_EVENT_HANDLER:nagios_type}: %
{DATA:nagios_hostname};%{DATA:nagios_service};%{DATA:nagios_state};%
{DATA:nagios_statelevel};%{DATA:nagios_event_handler_name}
NAGIOS_HOST_EVENT_HANDLER %{NAGIOS_TYPE_HOST_EVENT_HANDLER:nagios_type}: %
{DATA:nagios_hostname};%{DATA:nagios_state};%{DATA:nagios_statelevel};%
{DATA:nagios_event_handler_name} NAGIOS_TIMEPERIOD_TRANSITION %
{NAGIOS_TYPE_TIMEPERIOD_TRANSITION:nagios_type}: %{DATA:nagios_service};%
{DATA:nagios_unknown1};%{DATA:nagios_unknown2} ###################### #### External
checks ##################### #Disable host & service check
NAGIOS_EC_LINE_DISABLE_SVC_CHECK %{NAGIOS_TYPE_EXTERNAL_COMMAND:nagios_type}: %
{NAGIOS_EC_DISABLE_SVC_CHECK:nagios_command};%{DATA:nagios_hostname};%
{DATA:nagios_service} NAGIOS_EC_LINE_DISABLE_HOST_CHECK %
{NAGIOS_TYPE_EXTERNAL_COMMAND:nagios_type}: %
{NAGIOS_EC_DISABLE_HOST_CHECK:nagios_command};%{DATA:nagios_hostname} #Enable host
& service check NAGIOS_EC_LINE_ENABLE_SVC_CHECK %
{NAGIOS_TYPE_EXTERNAL_COMMAND:nagios_type}: %
{NAGIOS_EC_ENABLE_SVC_CHECK:nagios_command};%{DATA:nagios_hostname};%
{DATA:nagios_service} NAGIOS_EC_LINE_ENABLE_HOST_CHECK %
{NAGIOS_TYPE_EXTERNAL_COMMAND:nagios_type}: %
{NAGIOS_EC_ENABLE_HOST_CHECK:nagios_command};%{DATA:nagios_hostname} #Process host
& service check NAGIOS_EC_LINE_PROCESS_SERVICE_CHECK_RESULT %
{NAGIOS_TYPE_EXTERNAL_COMMAND:nagios_type}: %
{NAGIOS_EC_PROCESS_SERVICE_CHECK_RESULT:nagios_command};%{DATA:nagios_hostname};%
{DATA:nagios_service};%{DATA:nagios_state};%{GREEDYDATA:nagios_check_result}
NAGIOS_EC_LINE_PROCESS_HOST_CHECK_RESULT %
{NAGIOS_TYPE_EXTERNAL_COMMAND:nagios_type}: %
{NAGIOS_EC_PROCESS_HOST_CHECK_RESULT:nagios_command};%{DATA:nagios_hostname};%
{DATA:nagios_state};%{GREEDYDATA:nagios_check_result} #Disable host & service
notifications NAGIOS_EC_LINE_DISABLE_HOST_SVC_NOTIFICATIONS %
{NAGIOS_TYPE_EXTERNAL_COMMAND:nagios_type}: %
{NAGIOS_EC_DISABLE_HOST_SVC_NOTIFICATIONS:nagios_command};%
{GREEDYDATA:nagios_hostname} NAGIOS_EC_LINE_DISABLE_HOST_NOTIFICATIONS %
{NAGIOS_TYPE_EXTERNAL_COMMAND:nagios_type}: %
{NAGIOS_EC_DISABLE_HOST_NOTIFICATIONS:nagios_command};%{GREEDYDATA:nagios_hostname}
NAGIOS_EC_LINE_DISABLE_SVC_NOTIFICATIONS %
{NAGIOS_TYPE_EXTERNAL_COMMAND:nagios_type}: %
{NAGIOS_EC_DISABLE_SVC_NOTIFICATIONS:nagios_command};%{DATA:nagios_hostname};%
{GREEDYDATA:nagios_service} #Enable host & service notifications
NAGIOS_EC_LINE_ENABLE_HOST_SVC_NOTIFICATIONS %
{NAGIOS_TYPE_EXTERNAL_COMMAND:nagios_type}: %
{NAGIOS_EC_ENABLE_HOST_SVC_NOTIFICATIONS:nagios_command};%
{GREEDYDATA:nagios_hostname} NAGIOS_EC_LINE_ENABLE_HOST_NOTIFICATIONS %
{NAGIOS_TYPE_EXTERNAL_COMMAND:nagios_type}: %
{NAGIOS_EC_ENABLE_HOST_NOTIFICATIONS:nagios_command};%{GREEDYDATA:nagios_hostname}
NAGIOS_EC_LINE_ENABLE_SVC_NOTIFICATIONS %
{NAGIOS_TYPE_EXTERNAL_COMMAND:nagios_type}: %
{NAGIOS_EC_ENABLE_SVC_NOTIFICATIONS:nagios_command};%{DATA:nagios_hostname};%
{GREEDYDATA:nagios_service} #Schedule host & service downtime
NAGIOS_EC_LINE_SCHEDULE_HOST_DOWNTIME %{NAGIOS_TYPE_EXTERNAL_COMMAND:nagios_type}:
%{NAGIOS_EC_SCHEDULE_HOST_DOWNTIME:nagios_command};%{DATA:nagios_hostname};%
{NUMBER:nagios_start_time};%{NUMBER:nagios_end_time};%{NUMBER:nagios_fixed};%
{NUMBER:nagios_trigger_id};%{NUMBER:nagios_duration};%{DATA:author};%{DATA:comment}
#End matching line NAGIOSLOGLINE %{NAGIOSTIME} (?:%{NAGIOS_WARNING}|%
{NAGIOS_CURRENT_SERVICE_STATE}|%{NAGIOS_CURRENT_HOST_STATE}|%
{NAGIOS_SERVICE_NOTIFICATION}|%{NAGIOS_HOST_NOTIFICATION}|%{NAGIOS_SERVICE_ALERT}|%
{NAGIOS_HOST_ALERT}|%{NAGIOS_SERVICE_FLAPPING_ALERT}|%
{NAGIOS_HOST_FLAPPING_ALERT}|%{NAGIOS_SERVICE_DOWNTIME_ALERT}|%
```

```
{NAGIOS_HOST_DOWNTIME_ALERT}|%{NAGIOS_PASSIVE_SERVICE_CHECK}|%
{NAGIOS_PASSIVE_HOST_CHECK}|%{NAGIOS_SERVICE_EVENT_HANDLER}|%
{NAGIOS_HOST_EVENT_HANDLER}|%{NAGIOS_TIMEPERIOD_TRANSITION}|%
{NAGIOS_EC_LINE_DISABLE_SVC_CHECK}|%{NAGIOS_EC_LINE_ENABLE_SVC_CHECK}|%
{NAGIOS_EC_LINE_DISABLE_HOST_CHECK}|%{NAGIOS_EC_LINE_ENABLE_HOST_CHECK}|%
{NAGIOS_EC_LINE_PROCESS_HOST_CHECK_RESULT}|%
{NAGIOS_EC_LINE_PROCESS_SERVICE_CHECK_RESULT}|%
{NAGIOS_EC_LINE_SCHEDULE_HOST_DOWNTIME}|%
{NAGIOS_EC_LINE_DISABLE_HOST_SVC_NOTIFICATIONS}|%
{NAGIOS_EC_LINE_ENABLE_HOST_SVC_NOTIFICATIONS}|%
{NAGIOS_EC_LINE_DISABLE_HOST_NOTIFICATIONS}|%
{NAGIOS_EC_LINE_ENABLE_HOST_NOTIFICATIONS}|%
{NAGIOS_EC_LINE_DISABLE_SVC_NOTIFICATIONS}|%
{NAGIOS_EC_LINE_ENABLE_SVC_NOTIFICATIONS}|%{GREEDYDATA:greedy_message})
```

Mongodb

```
MONGO_LOG %{SYSLOGTIMESTAMP:timestamp} \[%{WORD:component}\] %{GREEDYDATA:message}
MONGO_QUERY \{ (?<={ ).*(?= } ntoreturn:) \} MONGO_SLOWQUERY %{WORD} %
{MONGO_WORDDASH:database}\.%{MONGO_WORDDASH:collection} %{WORD}: %
{MONGO_QUERY:query} %{WORD}:%{NONNEGINT:ntoreturn} %{WORD}:%{NONNEGINT:ntoskip} %
{WORD}:%{NONNEGINT:nscanned}.*nreturned:%{NONNEGINT:nreturned}..+ (?<duration>[0-
9]+)ms MONGO_WORDDASH \b[\w-]+\b MONGO3_SEVERITY \w MONGO3_COMPONENT %{WORD}|-
MONGO3_LOG %{TIMESTAMP_ISO8601:timestamp} %{MONGO3_SEVERITY:severity} %
{MONGO3_COMPONENT:component}%{SPACE}(?:\[%{DATA:context}\])? %{GREEDYDATA:message}
```

Mcollective-patterns

```
# Remember, these can be multi-line events. MCOLLECTIVE ., \[%
{TIMESTAMP_ISO8601:timestamp} #%{POSINT:pid}\]%{SPACE}%{LOGLEVEL:event_level}
MCOLLECTIVEAUDIT %{TIMESTAMP_ISO8601:timestamp}:
```

Mcollective

```
MCOLLECTIVEAUDIT %{TIMESTAMP_ISO8601:timestamp}:
```

Lucidworks-enterprise

```
# Parses the Lucidworks Enterprise logs. Requires grok-patterns OLDLUCIDFIND %
{IPORHOST:clientip}(?:%{SPACE})(?:%{USER:ident}|-)(?:%{SPACE})(?:%{USER:auth}|-)
(?:%{SPACE})\[%{HTTPDATE:timestamp}\](?:%{SPACE})"%{WORD:verb}(?:%{SPACE})%
{DATA:rawrequest}(?: HTTP/%{NUMBER:httpversion})"(?:%{SPACE})%{NUMBER:response}(?:%
{SPACE})(?:%{NUMBER:bytes}|-)(?:%{SPACE})(?:%{NUMBER:timing}|-)
```

Linux-syslog

```
SYSLOG5424PRINTASCII [!-~]+ SYSLOGBASE2 (?:%{SYSLOGTIMESTAMP:timestamp}|%
{TIMESTAMP_ISO8601:timestamp8601}) (?:%{SYSLOGFACILITY} )?%{SYSLOGHOST:logsource}+
(?: %{SYSLOGPROG}:|) SYSLOGPAMSESSION %{SYSLOGBASE} (?=%{GREEDYDATA:message})%
{WORD:pam_module}\(%{DATA:pam_caller}\): session %{WORD:pam_session_state} for user
%{USERNAME:username}(?: by %{GREEDYDATA:pam_by})? CRON_ACTION [A-Z ]+ CRONLOG %
{SYSLOGBASE} \(%{USER:user}\) %{CRON_ACTION:action} \(%{DATA:message}\) SYSLOGLINE
%{SYSLOGBASE2} %{GREEDYDATA:message} # IETF 5424 syslog(8) format (see
http://www.rfc-editor.org/info/rfc5424) SYSLOG5424PRI <%{NONNEGINT:syslog5424_pri}>
SYSLOG5424SD \[%{DATA}\]+ SYSLOG5424BASE %{SYSLOG5424PRI}%
{NONNEGINT:syslog5424_ver} +(?:%{TIMESTAMP_ISO8601:syslog5424_ts}|-) +(?:%
{HOSTNAME:syslog5424_host}|-) +(-|%{SYSLOG5424PRINTASCII:syslog5424_app}) +(-|%
{SYSLOG5424PRINTASCII:syslog5424_proc}) +(-|%
{SYSLOG5424PRINTASCII:syslog5424_msgid}) +(?:%{SYSLOG5424SD:syslog5424_sd}|-|)
SYSLOG5424LINE %{SYSLOG5424BASE} +%{GREEDYDATA:syslog5424_msg}
```

Junos

```
# JUNOS 11.4 RT_FLOW patterns RT_FLOW_EVENT
(RT_FLOW_SESSION_CREATE|RT_FLOW_SESSION_CLOSE|RT_FLOW_SESSION_DENY) RT_FLOW1 %
{RT_FLOW_EVENT:event}: %{GREEDYDATA:close-reason}: %{IP:src-ip}/%{INT:src-port}->%
{IP:dst-ip}/%{INT:dst-port} %{DATA:service} %{IP:nat-src-ip}/%{INT:nat-src-port}->%
{IP:nat-dst-ip}/%{INT:nat-dst-port} %{DATA:src-nat-rule-name} %{DATA:dst-nat-rule-
name} %{INT:protocol-id} %{DATA:policy-name} %{DATA:from-zone} %{DATA:to-zone} %
{INT:session-id} \d+\(%{DATA:sent}\) \d+\(%{DATA:received}\) %{INT:elapsed-time} .*
RT_FLOW2 %{RT_FLOW_EVENT:event}: session created %{IP:src-ip}/%{INT:src-port}->%
{IP:dst-ip}/%{INT:dst-port} %{DATA:service} %{IP:nat-src-ip}/%{INT:nat-src-port}->%
{IP:nat-dst-ip}/%{INT:nat-dst-port} %{DATA:src-nat-rule-name} %{DATA:dst-nat-rule-
name} %{INT:protocol-id} %{DATA:policy-name} %{DATA:from-zone} %{DATA:to-zone} %
{INT:session-id} .* RT_FLOW3 %{RT_FLOW_EVENT:event}: session denied %{IP:src-ip}/%
{INT:src-port}->%{IP:dst-ip}/%{INT:dst-port} %{DATA:service} %{INT:protocol-id}\
(\d\) %{DATA:policy-name} %{DATA:from-zone} %{DATA:to-zone} .*
```

Java

```
JAVACLASS (?:[a-zA-Z$_][a-zA-Z$_0-9]*\.)*[a-zA-Z$_][a-zA-Z$_0-9]* #Space is an allowed character to match special cases like 'Native Method' or 'Unknown Source' JAVAFILE (?:[A-Za-z0-9_. -]+) #Allow special <init>, <clinit> methods JAVAMETHOD (?:(<(?:cl)?init>)|[a-zA-Z$_][a-zA-Z$_0-9]*) #Line number is optional in special cases 'Native method' or 'Unknown source' JAVASTACKTRACEPART %{SPACE}at %{JAVACLASS:class}\.%{JAVAMETHOD:method}\(%{JAVAFILE:file}(?::%{NUMBER:line})?\) # Java Logs JAVATHREAD (?:[A-Z]{2}-Processor[\d]+) JAVACLASS (?:[a-zA-Z0-9-]+\.)+[A-Za-z0-9$]+ JAVAFILE (?:[A-Za-z0-9_.-]+) JAVASTACKTRACEPART at %{JAVACLASS:class}\.%{WORD:method}\(%{JAVAFILE:file}:%{NUMBER:line}\) JAVALOGMESSAGE (.*) # MMM dd, yyyy HH:mm:ss eg: Jan 9, 2014 7:13:13 AM CATALINA_DATESTAMP %{MONTH} %{MONTHDAY}, 20%{YEAR} %{HOUR}:?%{MINUTE}(?::?%{SECOND}) (?:AM|PM) # yyyy-MM-dd HH:mm:ss,SSS ZZZ eg: 2014-01-09 17:32:25,527 -0800 TOMCAT_DATESTAMP 20%{YEAR}-%{MONTHNUM}-%{MONTHDAY} %{HOUR}:?%{MINUTE}(?::?%{SECOND}) %{ISO8601_TIMEZONE} CATALINALOG %{CATALINA_DATESTAMP:timestamp} %{JAVACLASS:class} %{JAVALOGMESSAGE:logmessage} # 2014-01-09 20:03:28,269 -0800 | ERROR | com.example.service.ExampleService - something compeletely unexpected happened... TOMCATLOG %{TOMCAT_DATESTAMP:timestamp} \| %{LOGLEVEL:level} \| %{JAVACLASS:class} - %{JAVALOGMESSAGE:logmessage}
```

Haproxy

```
## These patterns were tested w/ haproxy-1.4.15 ## Documentation of the haproxy log formats can be found at the following links: ## http://code.google.com/p/haproxy-docs/wiki/HTTPLogFormat ## http://code.google.com/p/haproxy-docs/wiki/TCPLogFormat HAPROXYTIME (?!<[0-9])%{HOUR:haproxy_hour}:%{MINUTE:haproxy_minute}(?::%{SECOND:haproxy_second})(?![0-9]) HAPROXYDATE %{MONTHDAY:haproxy_monthday}/%{MONTH:haproxy_month}/%{YEAR:haproxy_year}:%{HAPROXYTIME:haproxy_time}.%{INT:haproxy_milliseconds} # Override these default patterns to parse out what is captured in your haproxy.cfg HAPROXYCAPTUREDREQUESTHEADERS %{DATA:captured_request_headers} HAPROXYCAPTUREDRESPONSEHEADERS %{DATA:captured_response_headers} # Example: # These haproxy config lines will add data to the logs that are captured # by the patterns below. Place them in your custom patterns directory to # override the defaults. # # capture request header Host len 40 # capture request header X-Forwarded-For len 50 # capture request header Accept-Language len 50 # capture request header referrer len 200 # capture request header User-Agent len 200 # # capture response header Content-Type len 30 # capture response header Content-Encoding len 10 # capture response header Cache-Control len 200 # capture response header Last-Modified len 200 # # HAPROXYCAPTUREDREQUESTHEADERS %{DATA:request_header_host}\|%{DATA:request_header_x_forwarded_for}\|%{DATA:request_header_accept_language}\|%{DATA:request_header_referrer}\|%{DATA:request_header_user_agent} # HAPROXYCAPTUREDRESPONSEHEADERS %{DATA:response_header_content_type}\|%{DATA:response_header_content_encoding}\|%{DATA:response_header_cache_control}\|%{DATA:response_header_last_modified} # parse a haproxy 'httplog' line HAPROXYHTTPBASE %{IP:client_ip}:%{INT:client_port} \[%{HAPROXYDATE:accept_date}\] %{NOTSPACE:frontend_name} %{NOTSPACE:backend_name}/%{NOTSPACE:server_name} %{INT:time_request}/%{INT:time_queue}/%{INT:time_backend_connect}/%{INT:time_backend_response}/%{NOTSPACE:time_duration} %{INT:http_status_code} %{NOTSPACE:bytes_read} %{DATA:captured_request_cookie} %{DATA:captured_response_cookie} %{NOTSPACE:termination_state} %{INT:actconn}/%{INT:feconn}/%{INT:beconn}/%{INT:srvconn}/%{NOTSPACE:retries} %{INT:srv_queue}/%{INT:backend_queue} (\{%{HAPROXYCAPTUREDREQUESTHEADERS}\})?( )?(\{%{HAPROXYCAPTUREDRESPONSEHEADERS}\})?( )?"(<BADREQ>|(%{WORD:http_verb} (%{URIPROTO:http_proto}://)?(?:%{USER:http_user}(?::[^@]*)?@)?(?:%{URIHOST:http_host})?(?:%{URIPATHPARAM:http_request})?( HTTP/%{NUMBER:http_version})?))?" HAPROXYHTTP (?:%{SYSLOGTIMESTAMP:syslog_timestamp}|%{TIMESTAMP_ISO8601:timestamp8601}) %{IPORHOST:syslog_server} %{SYSLOGPROG}: %{HAPROXYHTTPBASE} # parse a haproxy 'tcplog' line HAPROXYTCP (?:%{SYSLOGTIMESTAMP:syslog_timestamp}|%{TIMESTAMP_ISO8601:timestamp8601}) %{IPORHOST:syslog_server} %{SYSLOGPROG}: %{IP:client_ip}:%{INT:client_port} \[%{HAPROXYDATE:accept_date}\] %{NOTSPACE:frontend_name} %{NOTSPACE:backend_name}/%{NOTSPACE:server_name} %{INT:time_queue}/%{INT:time_backend_connect}/%{NOTSPACE:time_duration} %{NOTSPACE:bytes_read} %{NOTSPACE:termination_state} %{INT:actconn}/%{INT:feconn}/%{INT:beconn}/%{INT:srvconn}/%{NOTSPACE:retries} %{INT:srv_queue}/%{INT:backend_queue}
```

Grok

```
USERNAME [a-zA-Z0-9._-]+ USER %{USERNAME} EMAILLOCALPART [a-zA-Z][a-zA-Z0-9_.+-=:]+
EMAILADDRESS %{EMAILLOCALPART}@%{HOSTNAME} HTTPDUSER %{EMAILADDRESS}|%{USER} INT
(?:[+-]?(?:[0-9]+)) BASE10NUM (?<![0-9.+-])(?>[+-]?(?:(?:[0-9]+(?:\.[0-9]+)?)|(?:\.
[0-9]+))) NUMBER (?:%{BASE10NUM}) BASE16NUM (?<![0-9A-Fa-f])(?:[+-]?(?:0x)?(?:[0-
9A-Fa-f]+)) BASE16FLOAT \b(?<![0-9A-Fa-f.])(?:[+-]?(?:0x)?(?:(?:[0-9A-Fa-f]+(?:\.
[0-9A-Fa-f]*)?)|(?:\.[0-9A-Fa-f]+)))\b POSINT \b(?:[1-9][0-9]*)\b NONNEGINT \b(?:
[0-9]+)\b WORD \b\w+\b NOTSPACE \S+ SPACE \s* DATA .*? GREEDYDATA .* QUOTEDSTRING
(?>(?<!\\)(?>"(?>\\.|[^\\"]+)+"|""|(?>'(?>\\.|[^\\']+)+')|''|(?>`(?>\\.|
[^\\`]+)+`)|``)) UUID [A-Fa-f0-9]{8}-(?:[A-Fa-f0-9]{4}-){3}[A-Fa-f0-9]{12} # URN,
allowing use of RFC 2141 section 2.3 reserved characters URN urn:[0-9A-Za-z][0-9A-
Za-z-]{0,31}:(?:%[0-9a-fA-F]{2}|[0-9A-Za-z()+,.:=@;$_!*'/?#-])+
```

Networking

```
MAC (?:%{CISCOMAC}|%{WINDOWSMAC}|%{COMMONMAC}) CISCOMAC (?:(?:[A-Fa-f0-9]{4}\.){2}
[A-Fa-f0-9]{4}) WINDOWSMAC (?:(?:[A-Fa-f0-9]{2}-){5}[A-Fa-f0-9]{2}) COMMONMAC (?:
(?:[A-Fa-f0-9]{2}:){5}[A-Fa-f0-9]{2}) IPV6 ((([0-9A-Fa-f]{1,4}:){7}([0-9A-Fa-f]
{1,4}|:))|(([0-9A-Fa-f]{1,4}:){6}(:[0-9A-Fa-f]{1,4}|((25[0-5]|2[0-4]\d|1\d\d|[1-9]?
\d)(\.(25[0-5]|2[0-4]\d|1\d\d|[1-9]?\d)){3})|:))|(([0-9A-Fa-f]{1,4}:){5}(((:[0-9A-
Fa-f]{1,4}){1,2})|:((25[0-5]|2[0-4]\d|1\d\d|[1-9]?\d)(\.(25[0-5]|2[0-4]\d|1\d\d|[1-
9]?\d)){3})|:))|(([0-9A-Fa-f]{1,4}:){4}(((:[0-9A-Fa-f]{1,4}){1,3})|((:[0-9A-Fa-f]
{1,4})?:((25[0-5]|2[0-4]\d|1\d\d|[1-9]?\d)(\.(25[0-5]|2[0-4]\d|1\d\d|[1-9]?\d))
{3}))|:))|(([0-9A-Fa-f]{1,4}:){3}(((:[0-9A-Fa-f]{1,4}){1,4})|((:[0-9A-Fa-f]{1,4})
{0,2}:((25[0-5]|2[0-4]\d|1\d\d|[1-9]?\d)(\.(25[0-5]|2[0-4]\d|1\d\d|[1-9]?\d))
{3}))|:))|(([0-9A-Fa-f]{1,4}:){2}(((:[0-9A-Fa-f]{1,4}){1,5})|((:[0-9A-Fa-f]{1,4})
{0,3}:((25[0-5]|2[0-4]\d|1\d\d|[1-9]?\d)(\.(25[0-5]|2[0-4]\d|1\d\d|[1-9]?\d))
{3}))|:))|(([0-9A-Fa-f]{1,4}:){1}(((:[0-9A-Fa-f]{1,4}){1,6})|((:[0-9A-Fa-f]{1,4})
{0,4}:((25[0-5]|2[0-4]\d|1\d\d|[1-9]?\d)(\.(25[0-5]|2[0-4]\d|1\d\d|[1-9]?\d))
{3}))|:))|(:(((:[0-9A-Fa-f]{1,4}){1,7})|((:[0-9A-Fa-f]{1,4}){0,5}:((25[0-5]|2[0-
4]\d|1\d\d|[1-9]?\d)(\.(25[0-5]|2[0-4]\d|1\d\d|[1-9]?\d)){3}))|:))(%.+)? IPV4 (?<!
[0-9])(?:(?:[0-1]?[0-9]{1,2}|2[0-4][0-9]|25[0-5])[.](?:[0-1]?[0-9]{1,2}|2[0-4][0-
9]|25[0-5])[.](?:[0-1]?[0-9]{1,2}|2[0-4][0-9]|25[0-5])[.](?:[0-1]?[0-9]{1,2}|2[0-4]
[0-9]|25[0-5]))(?![0-9]) IP (?:%{IPV6}|%{IPV4}) HOSTNAME \b(?:[0-9A-Za-z][0-9A-Za-
z-]{0,62})(?:\.(?:[0-9A-Za-z][0-9A-Za-z-]{0,62}))*(\.?|\b) IPORHOST (?:%{IP}|%
{HOSTNAME}) HOSTPORT %{IPORHOST}:%{POSINT} # paths PATH (?:%{UNIXPATH}|%{WINPATH})
UNIXPATH (/([\w_%!$@:.,+~-]+|\\.)*)+ TTY (?:/dev/(pts|tty([pq])?)(\w+)?/?(?:[0-
9]+)) WINPATH (?>[A-Za-z]+:|\\)(?:\\[^\\?*]*)+ URIPROTO [A-Za-z]+(\+[A-Za-z+]+)?
URIHOST %{IPORHOST}(?::%{POSINT:port})? # uripath comes loosely from RFC1738, but
mostly from what Firefox # does not turn into %XX URIPATH (?:/[A-Za-z0-9$.+!*'()
{},~:;=@#%&_\-]*)+ #URIPARAM \?(?:[A-Za-z0-9]+(?:=(?:[^&]*))?(?:&(?:[A-Za-z0-9]+
(?:=(?:[^&]*))?)?)*)? URIPARAM \?[A-Za-z0-9$.+!*'|(){},~@#%&/=:;_?\-\[\]<>]*
URIPATHPARAM %{URIPATH}(?:%{URIPARAM})? URI %{URIPROTO}://(?:%{USER}(?::[^@]*)?@)?
(?:%{URIHOST})?(?:%{URIPATHPARAM})? # Months: January, Feb, 3, 03, 12, December
MONTH \b(?:[Jj]an(?:uary|uar)?|[Ff]eb(?:ruary|ruar)?|[Mm](?:a|ä)?r(?:ch|z)?|
[Aa]pr(?:il)?|[Mm]a(?:y|i)?|[Jj]un(?:e|i)?|[Jj]ul(?:y)?|[Aa]ug(?:ust)?|
[Ss]ep(?:tember)?|[Oo](?:c|k)?t(?:ober)?|[Nn]ov(?:ember)?|[Dd]e(?:c|z)(?:ember)?)\b
MONTHNUM (?:0?[1-9]|1[0-2]) MONTHNUM2 (?:0[1-9]|1[0-2]) MONTHDAY (?:(?:0[1-9])|(?:
[12][0-9])|(?:3[01])|[1-9]) # Days: Monday, Tue, Thu, etc... DAY (?:Mon(?:day)?
|Tue(?:sday)?|Wed(?:nesday)?|Thu(?:rsday)?|Fri(?:day)?|Sat(?:urday)?|Sun(?:day)?) #
Years? YEAR (?>\d\d){1,2} HOUR (?:2[0123]|[01]?[0-9]) MINUTE (?:[0-5][0-9]) # '60'
is a leap second in most time standards and thus is valid. SECOND (?:(?:[0-5]?[0-
9]|60)(?:[:.,][0-9]+)?) TIME (?!<[0-9])%{HOUR}:%{MINUTE}(?::%{SECOND})(?![0-9]) #
datestamp is YYYY/MM/DD-HH:MM:SS.UUUU (or something like it) DATE_US %{MONTHNUM}
[/-]%{MONTHDAY}[/-]%{YEAR} DATE_EU %{MONTHDAY}[./-]%{MONTHNUM}[./-]%{YEAR}
ISO8601_TIMEZONE (?:Z|[+-]%{HOUR}(?::?%{MINUTE})) ISO8601_SECOND (?:%{SECOND}|60)
TIMESTAMP_ISO8601 %{YEAR}-%{MONTHNUM}-%{MONTHDAY}[T ]%{HOUR}:?%{MINUTE}(?::?%
{SECOND})?%{ISO8601_TIMEZONE}? DATE %{DATE_US}|%{DATE_EU} DATESTAMP %{DATE}[- ]%
{TIME} TZ (?:[APMCE][SD]T|UTC) DATESTAMP_RFC822 %{DAY} %{MONTH} %{MONTHDAY} %{YEAR}
%{TIME} %{TZ} DATESTAMP_RFC2822 %{DAY}, %{MONTHDAY} %{MONTH} %{YEAR} %{TIME} %
{ISO8601_TIMEZONE} DATESTAMP_OTHER %{DAY} %{MONTH} %{MONTHDAY} %{TIME} %{TZ} %
{YEAR} DATESTAMP_EVENTLOG %{YEAR}%{MONTHNUM2}%{MONTHDAY}%{HOUR}%{MINUTE}%{SECOND}
HTTPDERROR_DATE %{DAY} %{MONTH} %{MONTHDAY} %{TIME} %{YEAR} # Syslog Dates: Month
Day HH:MM:SS SYSLOGTIMESTAMP %{MONTH} +%{MONTHDAY} %{TIME} PROG [\x21-\x5a\x5c\x5e-
\x7e]+ SYSLOGPROG %{PROG:program}(?:\[%{POSINT:pid}\])? SYSLOGHOST %{IPORHOST}
SYSLOGFACILITY <%{NONNEGINT:facility}.%{NONNEGINT:priority}> HTTPDATE %{MONTHDAY}/%
{MONTH}/%{YEAR}:%{TIME} %{INT} # Shortcuts QS %{QUOTEDSTRING} # Log formats
SYSLOGBASE %{SYSLOGTIMESTAMP:timestamp} (?:%{SYSLOGFACILITY} )?%
{SYSLOGHOST:logsource} %{SYSLOGPROG}: COMMONAPACHELOG %{IPORHOST:clientip} %
{HTTPDUSER:ident} %{USER:auth} \[%{HTTPDATE:timestamp}\] "(?:%{WORD:verb} %
{NOTSPACE:request}(?: HTTP/%{NUMBER:httpversion})?|%{DATA:rawrequest})" %
{NUMBER:response} (?:%{NUMBER:bytes}|-) COMBINEDAPACHELOG %{COMMONAPACHELOG} %
{QS:referrer} %{QS:agent} HTTPD20_ERRORLOG \[%{HTTPDERROR_DATE:timestamp}\] \[%
{LOGLEVEL:loglevel}\] (?:\[client %{IPORHOST:clientip}\] ){0,1}%
{GREEDYDATA:errormsg} HTTPD24_ERRORLOG \[%{HTTPDERROR_DATE:timestamp}\] \[%
{WORD:module}:%{LOGLEVEL:loglevel}\] \[pid %{POSINT:pid}:tid %{NUMBER:tid}\]( \(%
{POSINT:proxy_errorcode}\)%{DATA:proxy_errormessage}:)?( \[client %
{IPORHOST:client}:%{POSINT:clientport}\])? %{DATA:errorcode}: %{GREEDYDATA:message}
```

```
HTTPD_ERRORLOG %{HTTPD20_ERRORLOG}|%{HTTPD24_ERRORLOG} # Log Levels LOGLEVEL
([Aa]lert|ALERT|[Tt]race|TRACE|[Dd]ebug|DEBUG|[Nn]otice|NOTICE|[Ii]nfo|INFO|
[Ww]arn?(?:ing)?|WARN?(?:ING)?|[Ee]rr?(?:or)?|ERR?(?:OR)?|[Cc]rit?(?:ical)?|CRIT?
(?:ICAL)?|[Ff]atal|FATAL|[Ss]every|Severe|EMERG(?:ENCY)?|[Ee]merg(?:ency)?)
```

Firewalls

```
# NetScreen firewall logs NETSCREENSESSIONLOG %{SYSLOGTIMESTAMP:date} %
{IPORHOST:device} %{IPORHOST}: NetScreen device_id=%{WORD:device_id}%{DATA}:
start_time=%{QUOTEDSTRING:start_time} duration=%{INT:duration} policy_id=%
{INT:policy_id} service=%{DATA:service} proto=%{INT:proto} src zone=%
{WORD:src_zone} dst zone=%{WORD:dst_zone} action=%{WORD:action} sent=%{INT:sent}
rcvd=%{INT:rcvd} src=%{IPORHOST:src_ip} dst=%{IPORHOST:dst_ip} src_port=%
{INT:src_port} dst_port=%{INT:dst_port} src-xlated ip=%{IPORHOST:src_xlated_ip}
port=%{INT:src_xlated_port} dst-xlated ip=%{IPORHOST:dst_xlated_ip} port=%
{INT:dst_xlated_port} session_id=%{INT:session_id} reason=%{GREEDYDATA:reason} #==
Cisco ASA == CISCO_TAGGED_SYSLOG ^<%{POSINT:syslog_pri}>%{CISCOTIMESTAMP:timestamp}
( %{SYSLOGHOST:sysloghost})? ?: %%{CISCOTAG:ciscotag}: CISCOTIMESTAMP %{MONTH} +%
{MONTHDAY}(?: %{YEAR})? %{TIME} CISCOTAG [A-Z0-9]+-%{INT}-(?:[A-Z0-9_]+) # Common
Particles CISCO_ACTION Built|Teardown|Deny|Denied|denied|requested|permitted|denied
by ACL|discarded|est-allowed|Dropping|created|deleted CISCO_REASON Duplicate TCP
SYN|Failed to locate egress interface|Invalid transport field|No matching
connection|DNS Response|DNS Query|(?:%{WORD}\s*)* CISCO_DIRECTION
Inbound|inbound|Outbound|outbound CISCO_INTERVAL first hit|%{INT}-second interval
CISCO_XLATE_TYPE static|dynamic # ASA-1-104001 CISCOFW104001 \
((?:Primary|Secondary)\) Switching to ACTIVE - %{GREEDYDATA:switch_reason} # ASA-1-
104002 CISCOFW104002 \((?:Primary|Secondary)\) Switching to STANDBY - %
{GREEDYDATA:switch_reason} # ASA-1-104003 CISCOFW104003 \((?:Primary|Secondary)\)
Switching to FAILED\. # ASA-1-104004 CISCOFW104004 \((?:Primary|Secondary)\)
Switching to OK\. # ASA-1-105003 CISCOFW105003 \((?:Primary|Secondary)\) Monitoring
on [Ii]nterface %{GREEDYDATA:interface_name} waiting # ASA-1-105004 CISCOFW105004 \
((?:Primary|Secondary)\) Monitoring on [Ii]nterface %{GREEDYDATA:interface_name}
normal # ASA-1-105005 CISCOFW105005 \((?:Primary|Secondary)\) Lost Failover
communications with mate on [Ii]nterface %{GREEDYDATA:interface_name} # ASA-1-
105008 CISCOFW105008 \((?:Primary|Secondary)\) Testing [Ii]nterface %
{GREEDYDATA:interface_name} # ASA-1-105009 CISCOFW105009 \((?:Primary|Secondary)\)
Testing on [Ii]nterface %{GREEDYDATA:interface_name} (?:Passed|Failed) # ASA-2-
106001 CISCOFW106001 %{CISCO_DIRECTION:direction} %{WORD:protocol} connection %
{CISCO_ACTION:action} from %{IP:src_ip}/%{INT:src_port} to %{IP:dst_ip}/%
{INT:dst_port} flags %{GREEDYDATA:tcp_flags} on interface %{GREEDYDATA:interface} #
ASA-2-106006, ASA-2-106007, ASA-2-106010 CISCOFW106006_106007_106010 %
{CISCO_ACTION:action} %{CISCO_DIRECTION:direction} %{WORD:protocol} (?:from|src) %
{IP:src_ip}/%{INT:src_port}(\(%{DATA:src_fwuser}\))? (?:to|dst) %{IP:dst_ip}/%
{INT:dst_port}(\(%{DATA:dst_fwuser}\))? (?:on interface %{DATA:interface}|due to %
{CISCO_REASON:reason}) # ASA-3-106014 CISCOFW106014 %{CISCO_ACTION:action} %
{CISCO_DIRECTION:direction} %{WORD:protocol} src %{DATA:src_interface}:%{IP:src_ip}
(\(%{DATA:src_fwuser}\))? dst %{DATA:dst_interface}:%{IP:dst_ip}(\(%
{DATA:dst_fwuser}\))? \(type %{INT:icmp_type}, code %{INT:icmp_code}\) # ASA-6-
106015 CISCOFW106015 %{CISCO_ACTION:action} %{WORD:protocol} \(%{DATA:policy_id}\)
from %{IP:src_ip}/%{INT:src_port} to %{IP:dst_ip}/%{INT:dst_port} flags %
{DATA:tcp_flags} on interface %{GREEDYDATA:interface} # ASA-1-106021 CISCOFW106021
%{CISCO_ACTION:action} %{WORD:protocol} reverse path check from %{IP:src_ip} to %
{IP:dst_ip} on interface %{GREEDYDATA:interface} # ASA-4-106023 CISCOFW106023 %
{CISCO_ACTION:action}( protocol)? %{WORD:protocol} src %{DATA:src_interface}:%
{DATA:src_ip}(/%{INT:src_port})?(\(%{DATA:src_fwuser}\))? dst %
{DATA:dst_interface}:%{DATA:dst_ip}(/%{INT:dst_port})?(\(%{DATA:dst_fwuser}\))?( \
(type %{INT:icmp_type}, code %{INT:icmp_code}\))? by access-group "?%
{DATA:policy_id}"? \[%{DATA:hashcode1}, %{DATA:hashcode2}\] # ASA-4-106100, ASA-4-
106102, ASA-4-106103 CISCOFW106100_2_3 access-list %{NOTSPACE:policy_id} %
{CISCO_ACTION:action} %{WORD:protocol} for user '%{DATA:src_fwuser}' %
{DATA:src_interface}/%{IP:src_ip}\(%{INT:src_port}\) -> %{DATA:dst_interface}/%
{IP:dst_ip}\(%{INT:dst_port}\) hit-cnt %{INT:hit_count} %{CISCO_INTERVAL:interval}
\[%{DATA:hashcode1}, %{DATA:hashcode2}\] # ASA-5-106100 CISCOFW106100 access-list %
{NOTSPACE:policy_id} %{CISCO_ACTION:action} %{WORD:protocol} %
{DATA:src_interface}/%{IP:src_ip}\(%{INT:src_port}\)(\(%{DATA:src_fwuser}\))? -> %
{DATA:dst_interface}/%{IP:dst_ip}\(%{INT:dst_port}\)(\(%{DATA:src_fwuser}\))? hit-
cnt %{INT:hit_count} %{CISCO_INTERVAL:interval} \[%{DATA:hashcode1}, %
{DATA:hashcode2}\] # ASA-6-110002 CISCOFW110002 %{CISCO_REASON:reason} for %
{WORD:protocol} from %{DATA:src_interface}:%{IP:src_ip}/%{INT:src_port} to %
{IP:dst_ip}/%{INT:dst_port} # ASA-6-302010 CISCOFW302010 %{INT:connection_count} in
```

```
use, %{INT:connection_count_max} most used # ASA-6-302013, ASA-6-302014, ASA-6-
302015, ASA-6-302016 CISCOFW302013_302014_302015_302016 %{CISCO_ACTION:action}(?: %
{CISCO_DIRECTION:direction})? %{WORD:protocol} connection %{INT:connection_id} for
%{DATA:src_interface}:%{IP:src_ip}/%{INT:src_port}( \(%{IP:src_mapped_ip}/%
{INT:src_mapped_port}\))?(\(%{DATA:src_fwuser}\))? to %{DATA:dst_interface}:%
{IP:dst_ip}/%{INT:dst_port}( \(%{IP:dst_mapped_ip}/%{INT:dst_mapped_port}\))?(\(%
{DATA:dst_fwuser}\))?( duration %{TIME:duration} bytes %{INT:bytes})?(?: %
{CISCO_REASON:reason})?( \(%{DATA:user}\))? # ASA-6-302020, ASA-6-302021
CISCOFW302020_302021 %{CISCO_ACTION:action}(?: %{CISCO_DIRECTION:direction})? %
{WORD:protocol} connection for faddr %{IP:dst_ip}/%{INT:icmp_seq_num}(?:\(%
{DATA:fwuser}\))? gaddr %{IP:src_xlated_ip}/%{INT:icmp_code_xlated} laddr %
{IP:src_ip}/%{INT:icmp_code}( \(%{DATA:user}\))? # ASA-6-305011 CISCOFW305011 %
{CISCO_ACTION:action} %{CISCO_XLATE_TYPE:xlate_type} %{WORD:protocol} translation
from %{DATA:src_interface}:%{IP:src_ip}(/%{INT:src_port})?(\(%{DATA:src_fwuser}\))?
to %{DATA:src_xlated_interface}:%{IP:src_xlated_ip}/%{DATA:src_xlated_port} # ASA-
3-313001, ASA-3-313004, ASA-3-313008 CISCOFW313001_313004_313008 %
{CISCO_ACTION:action} %{WORD:protocol} type=%{INT:icmp_type}, code=%{INT:icmp_code}
from %{IP:src_ip} on interface %{DATA:interface}( to %{IP:dst_ip})? # ASA-4-313005
CISCOFW313005 %{CISCO_REASON:reason} for %{WORD:protocol} error message: %
{WORD:err_protocol} src %{DATA:err_src_interface}:%{IP:err_src_ip}(\(%
{DATA:err_src_fwuser}\))? dst %{DATA:err_dst_interface}:%{IP:err_dst_ip}(\(%
{DATA:err_dst_fwuser}\))? \(type %{INT:err_icmp_type}, code %{INT:err_icmp_code}\)
on %{DATA:interface} interface\. Original IP payload: %{WORD:protocol} src %
{IP:orig_src_ip}/%{INT:orig_src_port}(\(%{DATA:orig_src_fwuser}\))? dst %
{IP:orig_dst_ip}/%{INT:orig_dst_port}(\(%{DATA:orig_dst_fwuser}\))? # ASA-5-321001
CISCOFW321001 Resource '%{WORD:resource_name}' limit of %{POSINT:resource_limit}
reached for system # ASA-4-402117 CISCOFW402117 %{WORD:protocol}: Received a non-
IPSec packet \(protocol= %{WORD:orig_protocol}\) from %{IP:src_ip} to %{IP:dst_ip}
# ASA-4-402119 CISCOFW402119 %{WORD:protocol}: Received an %{WORD:orig_protocol}
packet \(SPI= %{DATA:spi}, sequence number= %{DATA:seq_num}\) from %{IP:src_ip} \
(user= %{DATA:user}\) to %{IP:dst_ip} that failed anti-replay checking # ASA-4-
419001 CISCOFW419001 %{CISCO_ACTION:action} %{WORD:protocol} packet from %
{DATA:src_interface}:%{IP:src_ip}/%{INT:src_port} to %{DATA:dst_interface}:%
{IP:dst_ip}/%{INT:dst_port}, reason: %{GREEDYDATA:reason} # ASA-4-419002
CISCOFW419002 %{CISCO_REASON:reason} from %{DATA:src_interface}:%{IP:src_ip}/%
{INT:src_port} to %{DATA:dst_interface}:%{IP:dst_ip}/%{INT:dst_port} with different
initial sequence number # ASA-4-500004 CISCOFW500004 %{CISCO_REASON:reason} for
protocol=%{WORD:protocol}, from %{IP:src_ip}/%{INT:src_port} to %{IP:dst_ip}/%
{INT:dst_port} # ASA-6-602303, ASA-6-602304 CISCOFW602303_602304 %{WORD:protocol}:
An %{CISCO_DIRECTION:direction} %{GREEDYDATA:tunnel_type} SA \(SPI= %{DATA:spi}\)
between %{IP:src_ip} and %{IP:dst_ip} \(user= %{DATA:user}\) has been %
{CISCO_ACTION:action} # ASA-7-710001, ASA-7-710002, ASA-7-710003, ASA-7-710005,
ASA-7-710006 CISCOFW710001_710002_710003_710005_710006 %{WORD:protocol}
(?:request|access) %{CISCO_ACTION:action} from %{IP:src_ip}/%{INT:src_port} to %
{DATA:dst_interface}:%{IP:dst_ip}/%{INT:dst_port} # ASA-6-713172 CISCOFW713172
Group = %{GREEDYDATA:group}, IP = %{IP:src_ip}, Automatic NAT Detection
Status:\s+Remote end\s*%{DATA:is_remote_natted}\s*behind a NAT
device\s+This\s+end\s*%{DATA:is_local_natted}\s*behind a NAT device # ASA-4-733100
CISCOFW733100 \[\s*%{DATA:drop_type}\s*\] drop %{DATA:drop_rate_id} exceeded.
Current burst rate is %{INT:drop_rate_current_burst} per second, max configured
rate is %{INT:drop_rate_max_burst}; Current average rate is %
{INT:drop_rate_current_avg} per second, max configured rate is %
{INT:drop_rate_max_avg}; Cumulative total count is %{INT:drop_total_count} #== End
Cisco ASA == # Shorewall firewall logs SHOREWALL (%{SYSLOGTIMESTAMP:timestamp}) (%
{WORD:nf_host}) kernel:.*Shorewall:(%{WORD:nf_action1})?:(%{WORD:nf_action2})?.*IN=
(%{USERNAME:nf_in_interface})?.*(OUT= *MAC=(%{COMMONMAC:nf_dst_mac}):(%
{COMMONMAC:nf_src_mac})?|OUT=%{USERNAME:nf_out_interface}).*SRC=(%
{IPV4:nf_src_ip}).*DST=(%{IPV4:nf_dst_ip}).*LEN=(%{WORD:nf_len}).?*TOS=(%
{WORD:nf_tos}).?*PREC=(%{WORD:nf_prec}).?*TTL=(%{INT:nf_ttl}).?*ID=(%{INT:nf_id}).?
*PROTO=(%{WORD:nf_protocol}).?*SPT=(%{INT:nf_src_port}?.*DPT=%{INT:nf_dst_port}?.*)
#== End Shorewall
```

Exim

```
EXIM_MSGID [0-9A-Za-z]{6}-[0-9A-Za-z]{6}-[0-9A-Za-z]{2} EXIM_FLAGS (<=|[-=>*]>|[*]{2}|==) EXIM_DATE %{YEAR:exim_year}-%{MONTHNUM:exim_month}-%{MONTHDAY:exim_day} %{TIME:exim_time} EXIM_PID \[%{POSINT}\] EXIM_QT ((\d+y)?(\d+w)?(\d+d)?(\d+h)?(\d+m)?(\d+s)?) EXIM_EXCLUDE_TERMS (Message is frozen|(Start|End) queue run| Warning: | retry time not reached | no (IP address|host name) found for (IP address|host) | unexpected disconnection while reading SMTP command | no immediate delivery: |another process is handling this message) EXIM_REMOTE_HOST (H=(%{NOTSPACE:remote_hostname} )?(\(%{NOTSPACE:remote_heloname}\) )?\[%{IP:remote_host}\]) EXIM_INTERFACE (I=\[%{IP:exim_interface}\](:%{NUMBER:exim_interface_port})) EXIM_PROTOCOL (P=%{NOTSPACE:protocol}) EXIM_MSG_SIZE (S=%{NUMBER:exim_msg_size}) EXIM_HEADER_ID (id=%{NOTSPACE:exim_header_id}) EXIM_SUBJECT (T=%{QS:exim_subject})
```

Bro

```
# https://www.bro.org/sphinx/script-reference/log-files.html # http.log BRO_HTTP %{NUMBER:ts}\t%{NOTSPACE:uid}\t%{IP:orig_h}\t%{INT:orig_p}\t%{IP:resp_h}\t%{INT:resp_p}\t%{INT:trans_depth}\t%{GREEDYDATA:method}\t%{GREEDYDATA:domain}\t%{GREEDYDATA:uri}\t%{GREEDYDATA:referrer}\t%{GREEDYDATA:user_agent}\t%{NUMBER:request_body_len}\t%{NUMBER:response_body_len}\t%{GREEDYDATA:status_code}\t%{GREEDYDATA:status_msg}\t%{GREEDYDATA:info_code}\t%{GREEDYDATA:info_msg}\t%{GREEDYDATA:filename}\t%{GREEDYDATA:bro_tags}\t%{GREEDYDATA:username}\t%{GREEDYDATA:password}\t%{GREEDYDATA:proxied}\t%{GREEDYDATA:orig_fuids}\t%{GREEDYDATA:orig_mime_types}\t%{GREEDYDATA:resp_fuids}\t%{GREEDYDATA:resp_mime_types} # dns.log BRO_DNS %{NUMBER:ts}\t%{NOTSPACE:uid}\t%{IP:orig_h}\t%{INT:orig_p}\t%{IP:resp_h}\t%{INT:resp_p}\t%{WORD:proto}\t%{INT:trans_id}\t%{GREEDYDATA:query}\t%{GREEDYDATA:qclass}\t%{GREEDYDATA:qclass_name}\t%{GREEDYDATA:qtype}\t%{GREEDYDATA:qtype_name}\t%{GREEDYDATA:rcode}\t%{GREEDYDATA:rcode_name}\t%{GREEDYDATA:AA}\t%{GREEDYDATA:TC}\t%{GREEDYDATA:RD}\t%{GREEDYDATA:RA}\t%{GREEDYDATA:Z}\t%{GREEDYDATA:answers}\t%{GREEDYDATA:TTLs}\t%{GREEDYDATA:rejected} # conn.log BRO_CONN %{NUMBER:ts}\t%{NOTSPACE:uid}\t%{IP:orig_h}\t%{INT:orig_p}\t%{IP:resp_h}\t%{INT:resp_p}\t%{WORD:proto}\t%{GREEDYDATA:service}\t%{NUMBER:duration}\t%{NUMBER:orig_bytes}\t%{NUMBER:resp_bytes}\t%{GREEDYDATA:conn_state}\t%{GREEDYDATA:local_orig}\t%{GREEDYDATA:missed_bytes}\t%{GREEDYDATA:history}\t%{GREEDYDATA:orig_pkts}\t%{GREEDYDATA:orig_ip_bytes}\t%{GREEDYDATA:resp_pkts}\t%{GREEDYDATA:resp_ip_bytes}\t%{GREEDYDATA:tunnel_parents} # files.log BRO_FILES %{NUMBER:ts}\t%{NOTSPACE:fuid}\t%{IP:tx_hosts}\t%{IP:rx_hosts}\t%{NOTSPACE:conn_uids}\t%{GREEDYDATA:source}\t%{GREEDYDATA:depth}\t%{GREEDYDATA:analyzers}\t%{GREEDYDATA:mime_type}\t%{GREEDYDATA:filename}\t%{GREEDYDATA:duration}\t%{GREEDYDATA:local_orig}\t%{GREEDYDATA:is_orig}\t%{GREEDYDATA:seen_bytes}\t%{GREEDYDATA:total_bytes}\t%{GREEDYDATA:missing_bytes}\t%{GREEDYDATA:overflow_bytes}\t%{GREEDYDATA:timedout}\t%{GREEDYDATA:parent_fuid}\t%{GREEDYDATA:md5}\t%{GREEDYDATA:sha1}\t%{GREEDYDATA:sha256}\t%{GREEDYDATA:extracted}
```

Bacula

```
BACULA_TIMESTAMP %{MONTHDAY}-%{MONTH} %{HOUR}:%{MINUTE} BACULA_HOST [a-zA-Z0-9-]+
BACULA_VOLUME %{USER} BACULA_DEVICE %{USER} BACULA_DEVICEPATH %{UNIXPATH}
BACULA_CAPACITY %{INT}{1,3}(,%{INT}{3})* BACULA_VERSION %{USER} BACULA_JOB %{USER}
BACULA_LOG_MAX_CAPACITY User defined maximum volume capacity %{BACULA_CAPACITY}
exceeded on device \"%{BACULA_DEVICE:device}\" \(%{BACULA_DEVICEPATH}\)
BACULA_LOG_END_VOLUME End of medium on Volume \"%{BACULA_VOLUME:volume}\" Bytes=%
{BACULA_CAPACITY} Blocks=%{BACULA_CAPACITY} at %{MONTHDAY}-%{MONTH}-%{YEAR} %
{HOUR}:%{MINUTE}. BACULA_LOG_NEW_VOLUME Created new Volume \"%
{BACULA_VOLUME:volume}\" in catalog. BACULA_LOG_NEW_LABEL Labeled new Volume \"%
{BACULA_VOLUME:volume}\" on device \"%{BACULA_DEVICE:device}\" \(%
{BACULA_DEVICEPATH}\). BACULA_LOG_WROTE_LABEL Wrote label to prelabeled Volume \"%
{BACULA_VOLUME:volume}\" on device \"%{BACULA_DEVICE}\" \(%{BACULA_DEVICEPATH}\)
BACULA_LOG_NEW_MOUNT New volume \"%{BACULA_VOLUME:volume}\" mounted on device \"%
{BACULA_DEVICE:device}\" \(%{BACULA_DEVICEPATH}\) at %{MONTHDAY}-%{MONTH}-%{YEAR} %
{HOUR}:%{MINUTE}. BACULA_LOG_NOOPEN \s+Cannot open %{DATA}: ERR=%
{GREEDYDATA:berror} BACULA_LOG_NOOPENDIR \s+Could not open directory %{DATA}: ERR=%
{GREEDYDATA:berror} BACULA_LOG_NOSTAT \s+Could not stat %{DATA}: ERR=%
{GREEDYDATA:berror} BACULA_LOG_NOJOBS There are no more Jobs associated with Volume
\"%{BACULA_VOLUME:volume}\". Marking it purged. BACULA_LOG_ALL_RECORDS_PRUNED All
records pruned from Volume \"%{BACULA_VOLUME:volume}\"; marking it \"Purged\"
BACULA_LOG_BEGIN_PRUNE_JOBS Begin pruning Jobs older than %{INT} month %{INT} days
. BACULA_LOG_BEGIN_PRUNE_FILES Begin pruning Files. BACULA_LOG_PRUNED_JOBS Pruned %
{INT} Jobs* for client %{BACULA_HOST:client} from catalog. BACULA_LOG_PRUNED_FILES
Pruned Files from %{INT} Jobs* for client %{BACULA_HOST:client} from catalog.
BACULA_LOG_ENDPRUNE End auto prune. BACULA_LOG_STARTJOB Start Backup JobId %{INT},
Job=%{BACULA_JOB:job} BACULA_LOG_STARTRESTORE Start Restore Job %{BACULA_JOB:job}
BACULA_LOG_USEDEVICE Using Device \"%{BACULA_DEVICE:device}\" BACULA_LOG_DIFF_FS
\s+%{UNIXPATH} is a different filesystem. Will not descend from %{UNIXPATH} into
it. BACULA_LOG_JOBEND Job write elapsed time = %{DATA:elapsed}, Transfer rate = %
{NUMBER} (K|M|G)? Bytes/second BACULA_LOG_NOPRUNE_JOBS No Jobs found to prune.
BACULA_LOG_NOPRUNE_FILES No Files found to prune. BACULA_LOG_VOLUME_PREVWRITTEN
Volume \"%{BACULA_VOLUME:volume}\" previously written, moving to end of data.
BACULA_LOG_READYAPPEND Ready to append to end of Volume \"%{BACULA_VOLUME:volume}\"
size=%{INT} BACULA_LOG_CANCELLING Cancelling duplicate JobId=%{INT}.
BACULA_LOG_MARKCANCEL JobId %{INT}, Job %{BACULA_JOB:job} marked to be canceled.
BACULA_LOG_CLIENT_RBJ shell command: run ClientRunBeforeJob \"%
{GREEDYDATA:runjob}\" BACULA_LOG_VSS (Generate )?VSS (Writer)? BACULA_LOG_MAXSTART
Fatal error: Job canceled because max start delay time exceeded.
BACULA_LOG_DUPLICATE Fatal error: JobId %{INT:duplicate} already running. Duplicate
job not allowed. BACULA_LOG_NOJOBSTAT Fatal error: No Job status returned from FD.
BACULA_LOG_FATAL_CONN Fatal error: bsock.c:133 Unable to connect to (Client: %
{BACULA_HOST:client}|Storage daemon) on %{HOSTNAME}:%{POSINT}. ERR=(?<berror>%
{GREEDYDATA}) BACULA_LOG_NO_CONNECT Warning: bsock.c:127 Could not connect to
(Client: %{BACULA_HOST:client}|Storage daemon) on %{HOSTNAME}:%{POSINT}. ERR=(?
<berror>%{GREEDYDATA}) BACULA_LOG_NO_AUTH Fatal error: Unable to authenticate with
File daemon at %{HOSTNAME}. Possible causes: BACULA_LOG_NOSUIT No prior or suitable
Full backup found in catalog. Doing FULL backup. BACULA_LOG_NOPRIOR No prior Full
backup Job record found. BACULA_LOG_JOB (Error: )?Bacula %{BACULA_HOST} %
{BACULA_VERSION} \(%{BACULA_VERSION}\): BACULA_LOGLINE %{BACULA_TIMESTAMP:bts} %
{BACULA_HOST:hostname} JobId %{INT:jobid}: (%{BACULA_LOG_MAX_CAPACITY}|%
{BACULA_LOG_END_VOLUME}|%{BACULA_LOG_NEW_VOLUME}|%{BACULA_LOG_NEW_LABEL}|%
{BACULA_LOG_WROTE_LABEL}|%{BACULA_LOG_NEW_MOUNT}|%{BACULA_LOG_NOOPEN}|%
{BACULA_LOG_NOOPENDIR}|%{BACULA_LOG_NOSTAT}|%{BACULA_LOG_NOJOBS}|%
{BACULA_LOG_ALL_RECORDS_PRUNED}|%{BACULA_LOG_BEGIN_PRUNE_JOBS}|%
{BACULA_LOG_BEGIN_PRUNE_FILES}|%{BACULA_LOG_PRUNED_JOBS}|%
{BACULA_LOG_PRUNED_FILES}|%{BACULA_LOG_ENDPRUNE}|%{BACULA_LOG_STARTJOB}|%
{BACULA_LOG_STARTRESTORE}|%{BACULA_LOG_USEDEVICE}|%{BACULA_LOG_DIFF_FS}|%
{BACULA_LOG_JOBEND}|%{BACULA_LOG_NOPRUNE_JOBS}|%{BACULA_LOG_NOPRUNE_FILES}|%
{BACULA_LOG_VOLUME_PREVWRITTEN}|%{BACULA_LOG_READYAPPEND}|%
{BACULA_LOG_CANCELLING}|%{BACULA_LOG_MARKCANCEL}|%{BACULA_LOG_CLIENT_RBJ}|%
{BACULA_LOG_VSS}|%{BACULA_LOG_MAXSTART}|%{BACULA_LOG_DUPLICATE}|%
```

```
{BACULA_LOG_NOJOBSTAT}|%{BACULA_LOG_FATAL_CONN}|%{BACULA_LOG_NO_CONNECT}|%
{BACULA_LOG_NO_AUTH}|%{BACULA_LOG_NOSUIT}|%{BACULA_LOG_JOB}|%{BACULA_LOG_NOPRIOR})
```

Aws

```
S3_REQUEST_LINE (?:%{WORD:verb} %{NOTSPACE:request}(?: HTTP/%{NUMBER:httpversion})?
|%{DATA:rawrequest}) S3_ACCESS_LOG %{WORD:owner} %{NOTSPACE:bucket} \[%
{HTTPDATE:timestamp}\] %{IP:clientip} %{NOTSPACE:requester} %{NOTSPACE:request_id}
%{NOTSPACE:operation} %{NOTSPACE:key} (?:"%{S3_REQUEST_LINE}"|-) (?:%
{INT:response:int}|-) (?:-|%{NOTSPACE:error_code}) (?:%{INT:bytes:int}|-) (?:%
{INT:object_size:int}|-) (?:%{INT:request_time_ms:int}|-) (?:%
{INT:turnaround_time_ms:int}|-) (?:%{QS:referrer}|-) (?:"?%{QS:agent}"?|-) (?:-|%
{NOTSPACE:version_id}) ELB_URIPATHPARAM %{URIPATH:path}(?:%{URIPARAM:params})?
ELB_URI %{URIPROTO:proto}://(?:%{USER}(?::[^@]*)?@)?(?:%{URIHOST:urihost})?(?:%
{ELB_URIPATHPARAM})? ELB_REQUEST_LINE (?:%{WORD:verb} %{ELB_URI:request}(?: HTTP/%
{NUMBER:httpversion})?|%{DATA:rawrequest}) ELB_ACCESS_LOG %
{TIMESTAMP_ISO8601:timestamp} %{NOTSPACE:elb} %{IP:clientip}:%{INT:clientport:int}
(?:(%{IP:backendip}:?:%{INT:backendport:int})|-) %
{NUMBER:request_processing_time:float} %{NUMBER:backend_processing_time:float} %
{NUMBER:response_processing_time:float} %{INT:response:int} %
{INT:backend_response:int} %{INT:received_bytes:int} %{INT:bytes:int} "%
{ELB_REQUEST_LINE}" See https://github.com/logstash-plugins/logstash-patterns-
core/tree/master/patterns
```

HTML Parser Stage

This parser stage processes the following HTML elements:

- `<title>`
- `<body>` (with tags removed)
- `<meta>`
- `<a>` and `<link>`

Additionally, you can configure JSoup selectors *(https://jsoup.org/cookbook/extracting-data/)* to extract specific HTML and CSS elements from a document and map them to PipelineDocument fields. For example, you could use this to process navigational `DIV` elements one way, then process content-ful `DIV` elements another way.

HTML Parser Stage supports jsoup selector methods by allowing the following special attribute values starting with `.` in `HtmlMappingRule`:

- `.text`: link:https://jsoup.org/apidocs/org/jsoup/nodes/Element.html#text%28%29
- `.ownText`: link:https://jsoup.org/apidocs/org/jsoup/nodes/Element.html#ownText%28%29
- `.html`: link:https://jsoup.org/apidocs/org/jsoup/nodes/Element.html#html%28%29
- `.outerHtml`: link:https://jsoup.org/apidocs/org/jsoup/nodes/Node.html#outerHtml%28%29
- `.data`: link:https://jsoup.org/apidocs/org/jsoup/nodes/Element.html#data%28%29

The HTML Transformation index pipeline stage is deprecated in favor of this parser stage.

HTML and CSS elements can be selected for extraction into new documents or fields:

- To create new documents from selected elements, configure `recordSelector`.
- To create new fields from selected elements, configure `mappings`.

Title, body, metadata, and links are only populated in the parent document. Both of these parameters support JSoup selectors *(https://jsoup.org/cookbook/extracting-data/)*, which provides a rich syntax for selecting HTML and CSS elements.

When entering configuration values in the UI, use *unescaped* characters, such as `\t` for the tab character. When entering configuration values in the API, use *escaped* characters, such as `\\t` for the tab character.

HTML content extraction

The Context Extraction setting in the HTML Parser has a special meaning.

By default, when unchecked or false, the HTML parser will essentially attempt to extract the text of the entire HTML page as the text that will be used in the Solr document.

However, when Content Extract is checked, or true, a set of heuristic rules are applied to attempt to automatically determine what node in the entire page is most likely to be the page content, based on the tree of nodes inside it and the text component of all the sub-nodes.

This may be helpful when you have a variety of different page formats, and also significant text on the pages which is not useful to add to the documents, such as significant text in the page header or footer.

However, since the algorithm is based on heuristics , it is possible that the results could change as the site is altered. When you need a high degree of certainty, we recommend that you use explicit rules of which nodes to extract to match your local configuration.

JSON Parser Stage

JSON parsing converts JSON content from a single document field into one or more new documents. This parser uses Solr's JsonRecordReader *(https://lucene.apache.org/solr/5_1_0/solr-solrj/org/apache/solr/common/util/JsonRecordReader.html)* to split JSON into sub-documents.

When entering configuration values in the UI, use *unescaped* characters, such as `\t` for the tab character. When entering configuration values in the API, use *escaped* characters, such as `\\t` for the tab character.

Solr Update Parser Stage

The Solr Update Parser stage parses Solr's "update" messages and transforms them into Fusion *PipelineDocuments*. This makes it possible to integrate with existing datasets that happen to be in Solr's "update" format and send them to an index pipeline.

This parser stage supports XML, JSON, CSV and JavaBin.

When parsing CSV files, consider using the *CSV parser*, which provides more options than the Solr Update Parser. All CSV files are treated the same in terms of structure and schema.

The `https://FUSION_HOST:FUSION_PORT/apps/solr-dist/example/exampledocs/` directory contains many of these "update" format documents. Fusion also ships with a `https://FUSION_HOST:FUSION_PORT/apps/solr-dist/bin/post` utility which is used to index documents from the file system. However, the `post` utility has no way of integrating with Fusion; its output is sent directly to Solr.

When entering configuration values in the UI, use *unescaped* characters, such as `\t` for the tab character. When entering configuration values in the API, use *escaped* characters, such as `\\t` for the tab character.

Text Parser Stage

The Plain Text parser can split a text file by lines or consume it into a single document.

Options for treatment of this filetype include:

- Plain Text Parser Fields
- Number of header rows to skip
- Split on line end or not
- Comment character
- Skip empty lines
- Charset

When entering configuration values in the UI, use *unescaped* characters, such as `\t` for the tab character. When entering configuration values in the API, use *escaped* characters, such as `\\t` for the tab character.

XML Parser Stage

The XML parser stage parses whole XML documents by default, but it can also be configured to parse only specific nodes without loading the entire document into memory. It can also split an XML document into multiple documents. XPATH-like expressions are used to select specific nodes to parse, such as `/posts/row` or `/posts/record`. Nested XML elements are flattened.

To create new documents from selected elements, configure `rootPaths`.

The XML Transformation index pipeline stage is deprecated in favor of this parser stage.
When entering configuration values in the UI, use *unescaped* characters, such as `\t` for the tab character. When entering configuration values in the API, use *escaped* characters, such as `\\t` for the tab character.

Configure Fusion for JWT

Fusion uses a shared secret key between the issuer and Fusion to encrypt the JWT payload.

How the JWT realm works

A JWT is comprised of three distinct parts: the header, payload, and signing key. Each of these parts are separately encoded using Base64url encoding.

- The **header** identifies the algorithm used to generate the token.
  Fusion uses both the HS256 and RS256 signature algorithms.
- The **payload** consists of data that will be passed with the token.
- The **signing key** validates the integrity of the token by using a "secret" to ensure the header and payload being submitted in the token match the header and payload stored in the signing key.

Within Fusion, the JWT realm uses an *authorization header* in the request to authenticate the user and the data inside the JWT token for the authorization. This authorization header uses the following format:

```
Bearer <jwt-token>
```

Upon receiving the authorization header, Fusion authenticates the token and emits a response accordingly.

Configure JWT for Fusion

Create a JWT token

Using the tool you use to validate users, create a JWT token. The token requires the following properties:

- `iss`: Issuer value. If the issuer value does not match the value configured in Fusion, the user will be denied access.
- `iat`: A JSON numeric date value. This value is calculated by counting the number of seconds between 1970-01-01T00:00:00Z UTC and the specified UTC date/time, ignoring leap seconds.
- `sub`: Subject. The name/id of the user. The user is logged in by this name.
- `groups`: The groups from the group-role mappings that this user belongs to. The groups key should match the one you specify while creating the JWT realm.

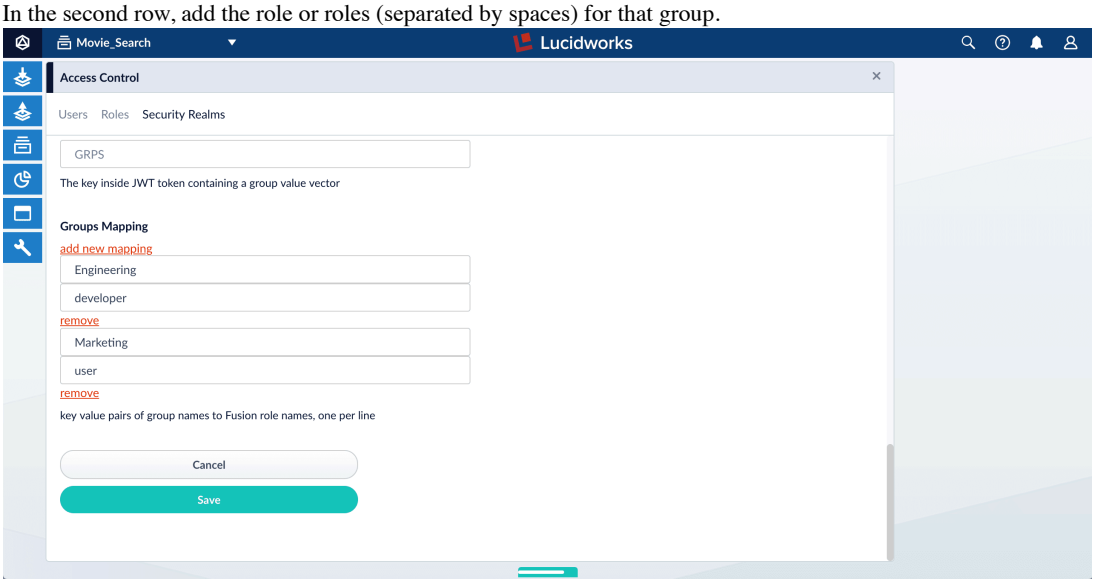
Example data inside token:

```
{
"iss":"fusion-enterprise-app",
"iat":"1562633069",
"sub":"username",
"groups": [“group-1”, “group-2”]
}
```

Create the Fusion JWT realm

To create a Fusion realm, in the Fusion UI:

1. Click **System > Access Control**.
2. Click the **Security Realms** tab, then **Add Security Realm**.
3. Enter a realm name. Under **type**, select **jwt**.
4. Create the security realm with the following values:

Value	Description
Realm type	jwt
Roles	Check the roles that this realm provides after successfully authenticating a user by default.
JWT Issuer	A unique value that is used in the JWT authorization header. This value should match the value of the `iss` in the token, for example `fusion-enterprise-app`.
JWT Set URI	*Optional*. The URI that the Signing Key can be downloaded from. Either this field or `Signing Key` must be filled.
Signing Key	*Optional*. A string of secret characters that will be used to encrypt the JWT token. The key must be a shared public key. Either this field or `JWT Set URI` must be filled.
Groups Key	The JWT token value that contains the list of groups the user is in.
Groups Mapping	A list of pairs which associate groups to roles. For example: `{<name of group>, <role assigned to group>}`. 1. Click **Add new mapping**. Two rows appear. 2. In the first row, add a group. The JWT token contains the groups for a particular user. 3. In the second row, add the role or roles (separated by spaces) for that group. 
User ID Attribute	The field in JWT from which Fusion takes the name of the user.

5. Click **Save**.

Validate the new realm

When you send a request to Fusion, you should receive a response. The request to Fusion looks like this:

```
curl http://127.0.0.1:8764/api/users -H 'authorization: Bearer <token-header>.
<token-payload>.<token-signing-key>'
```

`Bearer` is case sensitive.
If no signing key is used, truncate the JWT to read `… Bearer <token-header>.<token-payload>.`.

Resources

JWT.io *(https://jwt.io)* is an excellent resource for learning about JWT, creating tokens, debugging tokens, and more.

Example of generating a JWT using PyJWT

Typically, you will generate your JWT token from the application you are integrating with Fusion APIs.

This example, however, uses Python to create a JWT token with the PyJWT Python egg.

1. Install the PyJWT egg:

   ```
   pip install pyjwt
   ```

2. Inside a Python console, run the following command using your realm configurations:

   ```
   import jwt from datetime import datetime, timedelta key = '<signing key>'
   jwt_issuer = '<jwt issuer>' username = '<username>' groups = ['<groups assigned
   to user>'] iat = datetime.utcnow() exp = iat + timedelta(minutes=5) payload =
   {'iss': jwt_issuer, 'iat': iat, 'exp': exp,'sub': username, 'groups': groups}
   print (jwt.encode(payload, key, algorithm='HS256'))
   ```

   A Bearer token will be printed for use in API requests:

   ```
   eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpc3MiOiJmdXNpb24tYXBwIiwiaWF0IjoxNTYyNjM
   zMDY5LCJzdWIiOiJhZG1pbiIsImdyb3VwcyI6WyJncm91cDEiXX0.HYyv_XlijDeO1zdZ1Cd_gifgcIA
   FiZ1ldkbgk0hnWro
   ```

3. You can now use this as a bearer token in an authorization header to authenticate to Fusion.
   Test your Fusion API web service call with cURL. For example, if your role is an administrator, call:

   ```
   curl -H "Authorization: Bearer
   eyJhbGciOiJIUzI1NiJ9.eyJpc3MiOiJmdXNpb24iLCJpYXQiOjE1MzY3ODI1MTc4NjQsInN1YiI6InR
   lc3QtZnVzaW9uIiwiZ3JvdXBzIjpbImdyb3VwLW9uZSIsImdyb3VwLXR3byJdfQ._ACGk4q3Y5g8QgvL
   FUUQIcMN2ynACypcpei2JmYFlzc" http://localhost:8764/api/roles
   ```

Fusion Microservices

Fusion is comprised of microservices that drive features and functionality within a deployment. The services running in your deployment depend on the version of Fusion and the features you have enabled.

Get deployment details

You can view your deployment details using kubectl *(https://kubernetes.io/docs/reference/kubectl/)*.

1. First, verify that you have access to your cluster, replacing the example values with your parameters. This example is for a Fusion instance deployed using GKE.

```
gcloud container clusters get-credentials EXAMPLE-CLUSTER --region EXAMPLE-REGION --project EXAMPLE-PROJECT
```

2. Get the ports and services:

```
kubectl get svc -n EXAMPLE-NAME
```

3. Get the StatefulSets:

```
kubectl get statefulsets -n EXAMPLE-NAME
```

4. Get the deployments:

```
kubectl get deploy -n EXAMPLE-NAME
```

Overview

The table below lists the Fusion microservices deployed by the Helm chart. It also include Kubernetes services that manage traffic to microservices.

	Fusion is a complex distributed application composed of many stateful and stateless services designed to support demanding search-oriented workloads at high scale.

For Docker image versions associated with microservices, see the list of *Docker images and versions for each Fusion release*.

Microservice	Required for Fusion	Protocol	Deployment or StatefulSet	Node Pool Assignment	Autoscaling Supported	Description
`admin`	Yes	REST/HTTP	Deployment	system	Not required. Minimum of 1, but 2 pods are recommended for high availability.	Exposes endpoints for admin tasks, such as creating applications and running jobs.
`admin-ui`	No	Web	Deployment	system	Not required. One pod is enough for most clusters.	Serves static Web assets for the admin UI.
`argo`	No	HTTP	Deployment	system	Yes (CPU or custom metric).	Orchestrates parallel jobs on Kubernetes.
`argo-ui`	No	Web	Deployment	system	Not required. One pod is enough for most clusters.	Stores logs and prior Argo workflow runs.
`auth-ui`	No	Web	Deployment	system	Not required. One pod is enough for most clusters.	Serves static Web assets for the login form.
`connectors-classic`	No	REST/HTTP	StatefulSet	analytics or system	Not supported.	REST service for supporting non-RPC connector plugins. This microservice was previously named classic-rest-service.
`connectors`	No	REST/HTTP	Deployment	analytics or system	Not required. One pod is enough for most clusters.	Routes REST API requests to connectors-classic and connectors-rpc.
`connectors-backend`	No	gRPC	Deployment	analytics or system	Yes (CPU or custom metric).	gRPC service for managing SDK-based connector plugins.
`connector-plugin-<connector_plugin>`	No	HTTP/TCP	Deployment	analytics or system	Yes (CPU or custom metric).	Deployment for each connector plugin type. Note: There is a base deployment, connector-plugin with 0 replicas. This is used as a deployment template for each connector plugin type. It should not be deleted or scaled.
`devops-ui`	No	Web	Deployment	system	Not required. One pod is enough for most clusters.	Serves static Web assets for the DevOps UI. Available in Fusion 5.0 through 5.5.
`indexing`	Yes	REST/HTTP	Deployment	search or analytics depending on write-volume	Yes (CPU or custom metric).	Processes indexing requests.

Microservice	Required for Fusion	Protocol	Deployment or StatefulSet	Node Pool Assignment	Autoscaling Supported	Description
`insights`	No	Web	Deployment	system	Not required. One pod is enough for most clusters.	Serves the App Insights UI
`job-launcher`	No	REST/HTTP	Deployment	analytics	Not required. One pod is enough for most clusters.	Configures and launches the Spark driver pod for running Spark jobs
`job-rest-server`	No	REST/HTTP	Deployment	analytics	Not required. One pod is enough for most clusters.	Performs admin tasks for creating and running Spark jobs.
`jupyter`	No	HTTP	Deployment	analytics	Not required. One pod is enough for most clusters.	Jupyter notebook for ad hoc analytics and visualization.
`logstash`	No	HTTP	StatefulSet	system	Not required. Minimum of 1, but 2 pods are recommended for high availability.	Collects logs from the other microservices and either indexes into system_logs or ships them to an external service like Elastic
`milvus`	No	REST/HTTP	Deployment	analytics or system	Not required. One pod is enough for most clusters.	Dense Vector Search Engine for ML models active.
`milvus-mysql`	No	REST/HTTP	Deployment	analytics or system	Not required. One pod is enough for most clusters.	Handles metadata for Milvus service active.
`ml-model-service`	No	REST/HTTP and gRPC	Deployment	search	Yes (CPU or custom metric).	Exposes gRPC endpoints for generating predictions from ML models.
`pm-ui`	No	Web	Deployment	system	Not required. One pod is enough for most clusters.	Serves static Web assets for the Predictive Merchandiser app.
`proxy / api-gateway`	Yes	HTTP	Deployment	search	Not required. Minimum of 1, but 2 pods are recommended for high availability.	Performs authentication, authorization, and traffic routing.
`pulsar-bookkeeper`	Yes	HTTP	StatefulSet	search	At least 3 nodes in high availability. You need to run 3 or 5 to keep quorum.	Write Ahead Log (WAL) used for persistent message storage.

Microservice	Required for Fusion	Protocol	Deployment or StatefulSet	Node Pool Assignment	Autoscaling Supported	Description
`pulsar-broker`	Yes	HTTP and TCP	Deployment	search	At least 3 nodes in high availability.	Contains REST API for managing administration and dispatcher for handling all message transfers.
`query`	Yes	REST/HTTP	Deployment	search	Yes (CPU or custom metric).	Processes query requests.
`rules-ui`	No	Web	Deployment	system	Not required. One pod is enough for most clusters.	Serves static Web assets for the Rules UI.
`seldon-ambassador`	No	Web	Deployment	system	Not required. Minimum of 1, but 2 pods are recommended for high availability.	Load balancing and proxy for Seldon Core deployments.
`seldon-core`	No	REST/GRPC	Deployment	system	Yes (CPU or custom metric).	Serves models built in any model building framework.
`seldon-webhook-service`	No	Web	Deployment	system	Not required. One pod is enough for most clusters.	Maintains Seldon Core deployments for ML model serving active.
`solr`	Yes	HTTP	StatefulSet	At least 3 nodes in search, 2 in analytics, and 2 in system	Yes (CPU or custom metric).	Search engine.
`spark-driver`	No	n/a	single pod per job	analytics or dedicated Node Pool for Spark jobs	One per job.	Launched by the job-launcher to run a Spark job
`spark-executor`	No	n/a	one or more pods launched by the Spark driver for executing job tasks	analytics or dedicated Node Pool for Spark jobs	Depends on job configuration. Controlled by the spark.executor.instances setting.	Executes tasks for a Spark job
`sql-service`	No	REST/HTTP and JDBC	Deployment	analytics	Not required. One pod is enough for most clusters.	Performs admin tasks for creating and managing SQL catalog assets. Exposes a JDBC endpoint for the SQL service.

Microservice	Required for Fusion	Protocol	Deployment or StatefulSet	Node Pool Assignment	Autoscaling Supported	Description
`templating`	No	Web	Deployment	system	Not required. One pod is enough for most clusters.	Retrieves and renders Predictive Merchandiser templates.
`tikaserver`	No	HTTP	Deployment	system	Not required. One pod is enough for most clusters.	Is responsible for text extraction using Tika Server Parser
`webapps`	No	REST/HTTP	Deployment	system	Not required. One pod is enough for most clusters.	Serves App Studio-based Web apps.
`zookeeper`	Yes	TCP	StatefulSet	system	No. You need to run 1, 3, or 5 ZooKeeper pods to keep quorum. Do not use HPA for scaling ZooKeeper.	Stores centralized configuration and performs distributed coordination tasks.

Ports used by Fusion

Below you will find the list of pod ports for intra-cluster communications.

Service	Port
admin	8765
admin-ui	8080
auth-ui	8080
connector-plugin	9020, 5701
connectors	9010
connectors-backend	8771
connectors-classic	9000
devops-ui	8080
fusion-argo-argo-ui	2746
fusion-jupyter	8888
fusion-kafka	9092, 9093
fusion-headless	9092, 9093
fusion-ml-model-service-ambassador	80, 443
fusion-ml-model-service-mysql	3306
fusion-pulsar-bookkeeper	3181, 8000
fusion-pulsar-broker	8080, 6650
fusion-solr-exporter	8989
fusion-solr-headless	8983
fusion-solr-svc	8983
fusion-zookeeper	2181, 2281
fusion-zookeeper-headless	2181, 3888, 2888, 2281
indexing	8765
insights	8080
job-launcher	8083

Service	Port
`job-rest-server`	8081
`milvus`	19530, 19121
`ml-model-grpc`	6565
`ml-model-service`	8086
`moos-ui`	8080
`pm-ui`	8080
`prometheus-pushgateway`	9091
`proxy`	6764
`pulsar-broker`	8080, 6650
`query`	8787
`rules-ui`	8080
`seldon-webhook-service`	443
`sql-jdbc`	8088
`sql-service`	8085
`superset`	8088
`superset-proxy`	80
`templating`	5250
`tikaserver`	9998
`webapps`	8780

Custom JavaScript Query Stage examples

The following is a list of examples for reference when creating *custom JavaScript Query stages*.

Add a parameter to the query request

```
function(request,response , ctx, collection, solrServer, solrServerFactory) {
request.addParam("foo", "bar"); }
```

Add a parameter to the query response

This example contains a simple JavaScript function which copies information from the `ctx` (Context) object into the query response. Requirements:

- The `response` parameter cannot be used in query pipeline stages prior to the Solr query stage. This example assumes use in a later stage.
- The `response.initialEntity.appendStringList()` function only works if the query's `wt` (writer type) parameter is set to json or xml, for example: `wt=json`.

```
(function () { "use strict"; var List = Java.type('java.util.List'); function
add_to_response(key, list) { if (list.length > 0) {
response.initialEntity.appendStringList(key, Java.to(list, List)); } } return
function(request,response , ctx, collection, solrServer, solrServerFactory) {
add_to_response('banners', ctx.getProperty('banners')); add_to_response('landing-
pages', ctx.getProperty('redirects')); }; })();
```

Java object interactions

The JavaScript engine used by Fusion is the Nashorn engine from Oracle. See The Nashorn Java API *(https://docs.oracle.com/javase/8/docs/technotes/guides/scripting/nashorn/api.html)* for details.

The Nashorn engine exposes the full Java Classpath to JavaScript. This means that Java objects can be instantiated and called from JavaScript. Objects returned by a call to a Java function will not be JavaScript objects. For example, calling `typeof(object)` will not give meaningful results. Instead, you can use JavaScript to discover the Java type of an object. Once you know the Java type of an object, you can refer to the *Fusion Pipeline Javadocs* to learn more about the Java type and how to use Java Objects.

```
/* globals Java, logger*/ (function () { "use strict"; var Collection =
Java.type("java.util.Collection") var isDebug = false function logIfDebug(m) { if
(isDebug && m) logger.info(m, Array.prototype.slice.call(arguments).slice(1)); }
/** * @return a boolean indicating whether or not a given object looks like a Java
object */ function isJavaType(obj) { return (obj && typeof obj.getClass ===
'function' && typeof obj.notify === 'function' && typeof obj.hashCode ===
'function'); }; function isJavaArray(obj) { return (obj && isJavaType(obj) &&
getTypeOf(obj).endsWith('[]')); }; function isJavaCollection(collection) { return
isJavaType(collection) && Collection['class'].isInstance(collection); }; /** *
Nashorn allows obj.length and obj['property'] and obj.property functionality * for
some object types. This helps discover that fact at runtime */ function
isArrayLike(obj) { return isJavaCollection(obj) || Array.isArray(obj) ||
isJavaArray(obj); }; /** * For Java objects, return the simplified version of their
classname e.g. 'Format' for a java.text.Format * because Java naming conventions
call for upper case in the first character. * * For JavaScript objects, the type
will be a lower case string e.g. 'date' for a JavaScript Date. * Note: Nashorn uses
java.lang.String objects instead of JavaScript string objects */ function
getTypeOf(obj, verbose) { 'use strict'; var typ = 'unknown'; var isJava =
isJavaType(obj); logIfDebug("UTIL getTypeOf:called with verbose: ",verbose); //test
for java objects if (isJava && verbose) { typ = obj.getClass().getName();
logIfDebug("UTIL getTypeOf:inverbose block, type:: ",typ); } else if (isJava) { typ
= obj.getClass().getSimpleName(); } else if (obj === null) { typ = 'null'; } else
if (typeof (obj) === typeof (undefined)) { typ = 'undefined'; } else if (typeof
(obj) === typeof (String())) { typ = 'string'; } else if (Array.isArray(obj)) { typ
= 'array'; } else if (Object.prototype.toString.call(obj) === '[object Date]') {
typ = 'date'; } else { typ = obj ? typeof (obj) : typ; } return typ; }; return
function main(request, response, ctx, collection, solrServer, solrServerFactory) {
//output info about the objects passed in logger.info("request object has type
'{}'",getTypeOf(request,true)) //note, response is only valued after the solr query
stage logger.info("response object has type '{}'",getTypeOf(response,true))
logger.info("ctx object has type '{}'",getTypeOf(ctx,true)) logger.info("collection
object has type '{}'",getTypeOf(collection,true)) logger.info("solrServer object
has type '{}'",getTypeOf(solrServer,true)) logger.info("solrServerFactory object
has type '{}'",getTypeOf(solrServerFactory,true)) }; })();
```

For more information, see *JavaScript Query Stage Global Variables for Fusion 5.x.x*.

Docker Images by Fusion Version

The table below shows the Docker images and versions for each Fusion release.

To access this list through the command line, follow the instructions in *List All Docker Images and Versions*.

You can compare these Docker image versions to the *Fusion microservices* to better understand which containers make up which services.

	The following table is an example representation of a typical deployment. The Docker images may change, depending on the deployment's configuration.

Image	5.4.0	5.4.1	5.4.2	5.4.3	5.4.4	5.4.5
alpine	3.8	3.8	3.8	3.8	3.8	3.8
apache/tika	1.25	1.25	1.25	1.25	1.25	1.25
apachepulsar/pulsar-all	2.6.3	2.6.3	2.6.3	2.6.3	2.6.3	2.6.3
argoproj/argocli	v2.7.6	v2.7.6	v2.7.6	v2.7.6	v2.7.6	v2.7.6
argoproj/workflow-controller	v2.7.6	v2.7.6	v2.7.6	v2.7.6	v2.7.6	v2.7.6
bitnami/kubectl	*not specified*	*not specified*	*not specified*	*not specified*	*not specified*	*not specified*
bitnami/kubectl	1.15-debian-9	1.15-debian-9	1.15-debian-9	1.15-debian-9	1.15-debian-9	1.15-debian-9
bitnami/kubectl	latest	latest	latest	latest	latest	latest
busybox	*not specified*	*not specified*	*not specified*	*not specified*	*not specified*	*not specified*
busybox	1.31.1	1.31.1	1.31.1	1.31.1	1.31.1	1.31.1
docker.io/seldonio/seldon-core-operator	1.1.0	1.1.0	1.1.0	1.1.0	1.1.0	1.1.0
lucidworks/admin-ui	5.4.0	5.4.0	5.4.2	5.4.3	5.4.4	5.4.5
lucidworks/api-gateway	5.4.0	5.4.0	5.4.2	5.4.2	5.4.2	5.4.5
lucidworks/auth-ui	5.4.0	5.4.0	5.4.2	5.4.2	5.4.2	5.4.5
lucidworks/check-fusion-dependency	v1.2.0	v1.2.0	v1.2.0	v1.2.0	v1.2.0	v1.2.0
lucidworks/check-fusion-dependency	v1.3.0	v1.3.0	v1.3.0	v1.3.0	v1.3.0	v1.3.0
lucidworks/classic-rest-service	5.4.0	5.4.1	5.4.2	5.4.2	5.4.2	5.4.5
lucidworks/connectors-backend	5.4.0	5.4.1	5.4.2	5.4.2	5.4.2	5.4.5

Image	5.4.0	5.4.1	5.4.2	5.4.3	5.4.4	5.4.5
lucidworks/connectors	`5.4.0`	`5.4.1`	`5.4.2`	`5.4.2`	`5.4.2`	`5.4.5`
lucidworks/devops-ui	`5.4.0`	`5.4.0`	`5.4.2`	`5.4.2`	`5.4.2`	`5.4.5`
lucidworks/fusion-api	`5.4.0`	`5.4.0`	`5.4.2`	`5.4.2`	`5.4.2`	`5.4.5`
lucidworks/fusion-indexing	`5.4.0`	`5.4.0`	`5.4.2`	`5.4.2`	`5.4.2`	`5.4.5`
lucidworks/fusion-log-forwarder	`5.4.0`	`5.4.0`	`5.4.2`	`5.4.2`	`5.4.2`	`5.4.5`
lucidworks/insights	`5.4.0`	`5.4.0`	`5.4.2`	`5.4.2`	`5.4.2`	`5.4.5`
lucidworks/job-launcher	`5.4.0`	`5.4.0`	`5.4.2`	`5.4.2`	`5.4.2`	`5.4.5`
lucidworks/job-rest-server	`5.4.0`	`5.4.0`	`5.4.2`	`5.4.2`	`5.4.2`	`5.4.5`
lucidworks/keytool-utils	`v2.1.0`	`v2.1.0`	`v2.1.0`	`v2.1.0`	`v2.1.0`	`v2.1.0`
lucidworks/ml-model-service	`5.4.0`	`5.4.0`	`5.4.2`	`5.4.2`	`5.4.2`	`5.4.5`
lucidworks/pm-ui	`5.4.0`	`5.4.0`	`5.4.2`	`5.4.3`	`5.4.3`	`5.4.5`
lucidworks/query-pipeline	`5.4.0`	`5.4.0`	`5.4.2`	`5.4.2`	`5.4.2`	`5.4.5`
lucidworks/rules-ui	`5.4.0`	`5.4.0`	`5.4.2`	`5.4.3`	`5.4.3`	`5.4.5`
lucidworks/sql-service	`5.4.0`	`5.4.1`	`5.4.2`	`5.4.2`	`5.4.2`	`5.4.5`
lucidworks/templating	`5.4.0`	`5.4.0`	`5.4.2`	`5.4.2`	`5.4.2`	`5.4.5`
lucidworks/webapps	`5.4.0`	`5.4.0`	`5.4.2`	`5.4.2`	`5.4.2`	`5.4.5`
milvusdb/milvus	`0.10.2-cpu-d081520-8a2393`	`0.10.2-cpu-d081520-8a2393`	`0.10.2-cpu-d081520-8a2393`	`0.10.2-cpu-d081520-8a2393`	`0.10.2-cpu-d081520-8a2393`	`0.10.2-cpu-d081520-8a2393`

Image	5.4.0	5.4.1	5.4.2	5.4.3	5.4.4	5.4.5
mysql	`5.7.30`	`5.7.30`	`5.7.30`	`5.7.30`	`5.7.30`	`5.7.30`
quay.io/datawire/ambassador	`0.86.1`	`0.86.1`	`0.86.1`	`0.86.1`	`0.86.1`	`0.86.1`
solr	`8.8.2`	`8.8.2`	`8.8.2`	`8.8.2`	`8.8.2`	`8.8.2`
zookeeper	`3.6.2`	`3.6.2`	`3.6.2`	`3.6.2`	`3.6.2`	`3.6.2`

Gateway Configuration Properties

The Gateway configuration properties are generated during an application build.

The following table contains information about Fusion-related parameters.

name	type	Details
properties		
`audit.enabled`	java.lang.Boolean	`description`: Is audit enabled. `defaultValue`: true
`audit.excludes`	java.util.List<java.lang.String>	`description`: Comma separated list of paths to exclude from auditing.
`audit.get.enable`	java.lang.Boolean	`description`: Audit GET requests. `defaultValue`: false
`audit.get.excludes`	java.util.List<java.lang.String>	`description`: Comma separated list of paths to exclude from GET auditing. `defaultValue`: /api/ui-logs
`audit.get.resources-enabled`	java.lang.Boolean	`description`: Audit GET requests to static resources. `defaultValue`: false
`fusion.error.translation.enabled`	java.lang.Boolean	`defaultValue`: false
`fusion.error.translation.exclude-services`	java.lang.String[]	
`jwt.issuer.uri`	java.net.URI	
`jwt.token.cache-size`	java.lang.Integer	`description`: Size of the cache of valid JWT. If value exists in the cache, its signature is not validated. Cache is used for recognized, already logged users for audit logging. `defaultValue`: 1000
`jwt.token.expiration-secs`	java.lang.Integer	`description`: Validity time of JWT token, before next login will be required. `defaultValue`: 1800
`jwt.token.key`	java.lang.String	`description`: Name of the key in keystore. `defaultValue`: jwt-signer
`jwt.token.key-store`	java.lang.String	`description`: Path to keystore.
`jwt.token.pass`	java.lang.String	`description`: Password for the keystore. `defaultValue`:

name	type	Details
`security.auth.password-cache-size`	java.lang.Integer	`description`: Size of the cache of user credentials for native authentication. If 0, then cache is disabled. `defaultValue`: 0
`security.auth.url`	java.lang.String	`description`: Redirect path to authorization UI. `defaultValue`: /auth/login
`security.auth.user-cache-size`	java.lang.Integer	`description`: Size of the cache of user data. If 0, then cache is disabled. `defaultValue`: 100
`security.cors.allow-credentials`	java.lang.Boolean	
`security.cors.allowed-headers`	java.util.List<java.lang.String>	
`security.cors.allowed-methods`	java.util.List<java.lang.String>	
`security.cors.allowed-origins`	java.util.List<java.lang.String>	
`security.cors.enabled`	java.lang.Boolean	`defaultValue`: true
`security.internal.enforce-internal-address-exceptions`	java.util.List<java.lang.String>	`description`: List of names of service accounts which request IP does not check against the service registry. Applies only if enforceInternalAddressForServiceAccounts is true.
`security.internal.enforce-internal-address-for-service-accounts`	java.lang.Boolean	`description`: Check if service is listed in the service registry under IP or hostname of where the request originated. `defaultValue`: true
`security.internal.service-account-key`	java.lang.String	`description`: Key or password for all services to authenticate in proxy. `defaultValue`: invalid
`security.passthru`	java.util.List<java.lang.String>	`description`: List of paths to which anonymous access should be allowed.
`security.saml.max-authentication-age`	java.lang.Integer	`defaultValue`: 7200

name	type	Details

`fusion.cors.allow-credentials`	java.lang.Boolean	`deprecated`: true
		`deprecation`: {
		`replacement: security.cors.allow-credentials`
		}

`fusion.cors.allowed-headers`	java.util.List<java.lang.String>	`deprecated`: true
		`deprecation`: {
		`replacement: security.cors.allowed-headers`
		}

`fusion.cors.allowed-methods`	java.util.List<java.lang.String>	`deprecated`: true
		`deprecation`: {
		`replacement: security.cors.allowed-methods`
		}

`fusion.cors.allowed-methods`	java.util.List<java.lang.String>	`deprecated`: true
		`deprecation`: {
		`replacement: security.cors.allowed-methods`
		}

`fusion.cors.allowed-origins`	java.util.List<java.lang.String>	`deprecated`: true
		`deprecation`: {
		`replacement: security.cors.allow-origins`
		}

`fusion.cors.enabled`	java.lang.Boolean	`defaultValue`: true
		`deprecated`: true
		`deprecation`: {
		`replacement: security.cors.enabled`
		}

You can also enter `hints`: [ ] information in a similar format as the `properties` defined in the table.

Language Support in Fusion

The table below shows how various language are supported in the Fusion product suite. The meanings of the columns are as follows:

- **Encoding.** Fusion and App Studio support the language encodings of the selected languages both for search and display.
- **Language detection.** The superset of languages supported was determined based on our use of the Language Detection Library for Java that recognizes 71 languages and the supported languages in Solr.
- **Thesaurus.** Solr can support a supplied thesaurus for the selected language.
- **Did you mean (spell check).** Fusion can return spelling suggestions for the selected language.
- **Stemming.** Supported with an externally-supplied dictionary or out of the box with `HunspellStemFilter`.
- **Summary.** The ability to display matching text from a search request which may also include highlighting.
- **App Studio.** Text in the selected language can be displayed and Right-to-Left languages are also supported.
- **Administration tools.** The language is supported in the Fusion UI.
- **Backtranslation task.** Part of the *Data Augmentation Job*. Translates the input data into one or more intermediate languages before translating it back to the source language.
- **Synonym Substitution task.** Part of the *Data Augmentation Job*. Takes in the input text and substitutes some words with synonyms derived from the included wordner/ppdb dictionaries or user-supplied dictionaries.
- **Keystroke Misspelling task.** Part of the *Data Augmentation Job*. Simulates typos one might make based on the layout of the keyboard.
- **Split Word task.** Part of the *Data Augmentation Job*. Randomly splits words by introducing a space " " at some random point in the word.

Supported language	Encoding	Language detection	Thesaurus	Did you mean (spell check)	Stemming	Summary	App Studio	Administration tools	Backtranslation task	Synonym Substitution task	Keystroke Misspelling task	Split Word task
Afrikaans	x	x	x	x	x (hunspell)	x	x					
Albanian	x	x	x	x	x (hunspell)	x	x					
Arabic	x	x	x	x	x	x	x					
Aragonese	x	x	x	x	x (hunspell)	x	x					
Asturian	x	x	x	x	x (hunspell)	x	x					
Basque	x	x	x	x	x (hunspell)	x	x					
Belarusian	x	x	x	x	x (hunspell)	x	x					
Bengali	x	x	x	x	x (hunspell)	x	x					
Breton	x	x	x	x	x (hunspell)	x	x					
Brazilian Portuguese	x	x	x	x	x	x	x					
Bulgarian	x	x	x	x	x	x	x					
Catalan	x	x	x	x	x	x	x					
Chinese	x	x	x	x	x (performed as part of dictionary-based tokenization; separate stemming step not required)	x	x	x	x	x		
Croatian	x	x	x	x	x (hunspell)	x	x					
Czech	x	x	x	x	x	x	x					
Danish	x	x	x	x	x	x	x					

Supported language	Encoding	Language detection	Thesaurus	Did you mean (spell check)	Stemming	Summary	App Studio	Administration tools	Backtranslation task	Synonym Substitution task	Keystroke Misspelling task	Split Word task
Dutch	x	x	x	x	x	x	x	x	x	x	x	x
English	x	x	x	x	x	x	x	x	x	x	x	x
Estonian	x	x	x	x	x (hunspell)	x	x					
Finnish	x	x	x	x	x	x	x					
French	x	x	x	x	x	x	x	x	x	x	x	x
Galician	x	x	x	x	x	x	x					
German	x	x	x	x	x	x	x	x	x	x	x	x
Greek	x	x	x	x	x (hunspell)	x	x					
Gujarati	x	x	x	x	x (hunspell)	x	x					
Haitian	x	x	x	x	x (hunspell)	x	x					
Hebrew	x	x	x	x	x (hunspell)	x	x	x	x	x	x	
Hindi	x	x	x	x	x	x	x					
Hungarian	x	x	x	x	x (hunspell)	x	x					
Icelandic	x	x	x	x	x (hunspell)	x	x					
Indonesian	x	x	x	x	x	x	x					
Irish	x	x	x	x	x	x	x					
Italian	x	x	x	x	x	x	x	x	x	x	x	x
Japanese	x	x	x	x	x	x	x	x	x	x	x	

Supported language	Encoding	Language detection	Thesaurus	Did you mean (spell check)	Stemming	Summary	App Studio	Administration tools	Backtranslation task	Synonym Substitution task	Keystroke Misspelling task	Split Word task
Kannada	x	x	x	x	x (hunspell)	x	x					
Khmer	x	x	x	x	x (hunspell)	x	x					
Korean	x	x	x	x	x (hunspell)	x	x	x	x			
Lao	x	x	x	x		x	x					
Latvian	x	x	x	x	x	x	x					
Lithuanian	x	x	x	x	x (hunspell)	x	x					
Macedonian	x	x	x	x	x (hunspell)	x	x					
Malay	x	x	x	x	x (hunspell)	x	x					
Malayalam	x	x	x	x	x (hunspell)	x	x					
Maltese	x	x	x	x	x (hunspell)	x	x					
Marathi	x	x	x	x	x (hunspell)	x	x					
Myanmar				x		x	x					
Nepali	x	x	x	x	x (hunspell)	x	x					
Norwegian	x	x	x	x	x	x	x					
Occitan	x	x	x	x	x (hunspell)	x	x					
Persian	x	x	x	x	x (hunspell)	x	x					
Polish	x	x	x	x	x	x	x	x	x	x	x	x
Portuguese	x	x	x	x	x	x	x					

Supported language	Encoding	Language detection	Thesaurus	Did you mean (spell check)	Stemming	Summary	App Studio	Administration tools	Backtranslation task	Synonym Substitution task	Keystroke Misspelling task	Split Word task
Punjabi	x	x	x	x	x (hunspell)	x	x					
Romanian	x	x	x	x	x	x	x					
Russian	x	x	x	x	x	x	x					
Serbian	x	x	x	x	x (hunspell)	x	x					
Slovak	x	x	x	x	x (hunspell)	x	x					
Slovene	x	x	x	x	x (hunspell)	x	x					
Somali	x	x	x	x	x (hunspell)	x	x					
Spanish	x	x	x	x	x	x	x	x	x	x	x	x
Swahili	x	x	x	x	x (hunspell)	x	x					
Swedish	x	x	x	x	x	x	x					
Tagalog	x	x	x	x	x (hunspell)	x	x					
Tamil		x	x	x	x (hunspell)	x	x					
Telugu		x	x	x	x (hunspell)	x	x					
Thai	x	x	x	x	x (hunspell)	x	x					
Turkish	x	x	x	x	x	x	x					
Ukrainian	x	x	x	x	x	x	x	x	x		x	
Urdu		x	x	x	x (hunspell)	x	x					
Vietnamese		x	x	x	x (hunspell)	x	x					

Supported language	Encoding	Language detection	Thesaurus	Did you mean (spell check)	Stemming	Summary	App Studio	Administration tools	Backtranslation task	Synonym Substitution task	Keystroke Misspelling task	Split Word task
Walloon	x	x	x	x	x (hunspell)	x	x					
Welsh	x	x	x	x	x (hunspell)	x	x					
Yiddish	x	x	x	x	x (hunspell)	x	x					

Solr Reference Guide

See Solr Reference Guide *(https://solr.apache.org/guide/)* for all Solr guides.

Fusion release	Solr version
5.8.0 to latest	fusion-solr 5.8.0 (based on Solr 9.1.1 *(https://solr.apache.org/guide/9_1/)*)
5.7.0 to 5.7.1	fusion-solr 8.11.1 (based on Solr 8.11.1 *(https://solr.apache.org/guide/8_11/)*)
5.5.0 to 5.6.1	8.11.1 *(https://solr.apache.org/guide/8_11/)*
5.4.0 to 5.4.5	8.8.2 *(https://solr.apache.org/guide/8_8/)*
5.3.0 to 5.3.5	8.6.3 *(https://solr.apache.org/guide/8_6/)*
5.1.0 to 5.2.2	8.4.1 *(https://solr.apache.org/guide/8_4/)*
5.0.2	8.3.1 *(https://solr.apache.org/guide/8_3/)*
5.0.0 to 5.0.1	8.2.0 *(https://solr.apache.org/guide/8_2/)*

Spark Reference Guide

All Fusion 5 releases ship with Spark 2.4. See the Spark 2.4 reference guide *(https://spark.apache.org/docs/2.4.0/)*.

Developer Docs

These docs pertain to users of Lucidworks programmatic interfaces for development of front-end applications, custom connectors, custom pipeline stages, and SQL queries.

Fusion REST APIs

	When you install a connector or plug-in on Fusion 5.x.x, you do not need to supply the .zip file. The installation obtains the .zip file from the repository. Fusion 4.x.x and earlier releases required you to supply the .zip file.

How to access the API in Kubernetes

API endpoints for Kubernetes deployments use different paths than for on-prem deployments.

How to access individual API endpoints

```
https://FUSION_HOST:FUSION_PORT/SERVICE_NAME/ENDPOINT
```

Example: `https://FUSION_HOST:FUSION_PORT/query/query-pipelines` *(https://FUSION_HOST:FUSION_PORT/query/query-pipelines)*

How to access the Swagger UI

```
https://FUSION_HOST:FUSION_PORT/api/SERVICE_NAME/swagger-ui.html
```

How to access API specs (in JSON format)

```
https://FUSION_HOST:FUSION_PORT/SERVICE_NAME/v2/api-docs
```

Example: `https://FUSION_HOST:FUSION_PORT/query/v2/api-docs` *(https://FUSION_HOST:FUSION_PORT/query/v2/api-docs)*

How to access the on-prem API service

Fusion API services are designed to be accessed via Fusion's authentication proxy module which is part of the Fusion UI service. All applications should use this method to access the API service:

```
https://FUSION_HOST:FUSION_PORT/api/ENDPOINT
```

List all Fusion component services

The Fusion `introspect` endpoint lists basic information about endpoints and parameters for all Fusion endpoints, including the Connectors services endpoints:

```
curl -u USERNAME:PASSWORD https://FUSION_HOST:FUSION_PORT/api/introspect
```

Admin APIs

This group of APIs can help you with the general management of your Fusion instance. Use APIs to manage *App Studio* objects. Use the *Configurations* APIs to establish relationships between Fusion objects, connect your Fusion instance to *Solr clusters*, and manage license files. Or, create a *history* of service events, define schedules for *jobs*, and more.

Apps API

The Apps API manages your Fusion apps. To export or import an app, see the *Objects API*. You can also *import an app using the Fusion UI*.

Examples

List all configured apps

Input

```
curl -u USERNAME:PASSWORD https://FUSION_HOST:FUSION_PORT/api/apps
```

Output

```
[ { "id" : "1032724d-848a-4588-9503-c20702aa3a7b", "name" : "Test App",
"properties" : { "headerImageName" : "headerImage2", "tileColor" : "apps-red" } },
{ "id" : "7fc4d4e5-d3bd-4b21-b85e-9aa29c47b281", "name" : "Movie Search",
"properties" : { "headerImageName" : "headerImage3", "tileColor" : "apps-dark" } }
]
```

Get the configuration for a specific app

Input

```
curl -u USERNAME:PASSWORD https://FUSION_HOST:FUSION_PORT/api/apps/8fc4d4e5-d3bd-
4b21-b85e-9aa29c47b281
```

Output

```
{ "id" : "7fc4d4e5-d3bd-4b21-b85e-9aa29c47b281", "name" : "Movie Search",
"properties" : { "headerImageName" : "headerImage3", "tileColor" : "apps-dark" } }
```

Input

```
curl -u USERNAME:PASSWORD -X POST -H 'Content-type: application/json' -d '{"id":
"MyNewApp", "name":"My new app", "description":"A really great new app"}'
https://FUSION_HOST:FUSION_PORT/api/apps
```

Output

```
{ "id" : "MyNewApp", "name" : "My new app", "description" : "A really great new app" }
```

Blob Store API

The Blob Store REST API allows storing binary objects in Solr. The primary use case for this is to store entity extraction models, lookup lists or exclusion lists for use in *index pipelines*. This may include the entity extraction models and lookup lists included with Fusion in the `{fusion_path}/data/nlp` directory, or files that you have created on your own.

You can also work with blobs in the Fusion UI using the *blob manager*.

When you upload a blob without specifiying an app, the blob is stored in the `system_blobs` collection and you can access it via the *Object Explorer*.

	Previously, you could use the Blob Store API to install connectors. Instead, use the *Connectors API*.

Blob Types

A `resourceType` query parameter can be used to specify the blob type. For example, specify `file` when uploading a text or CSV file, like this:

```
curl -H 'Content-Type: text/csv' -u USERNAME:PASSWORD -X PUT
'https://FUSION_HOST:FUSION_PORT/api/apps/APP_NAME/blobs/quickstart/EXAMPLE.csv?
resourceType=file' --data-binary @/path/EXAMPLE.csv
```

The complete list of valid values for `resourceType` is below:

Type	Description
`banana`	A Banana *dashboard*
`catalog`	An *analytics catalog*
`driver:jdbc`	A *JDBC driver*
`file:js-index`	A JavaScript file for use with a *Managed Javascript index stage*
`file:js-query`	A JavaScript file for use with a *Managed Javascript query stage*
`file`	Any uploaded file, such as from the *Quickstart* or the *Index Workbench*
`model:open-nlp`	An *OpenNLP model*
`unspecified`	A blob of unknown type If no `resourceType` is specified on upload, "other" is assigned by default.
`plugin:index-stage`	An *index stage plugin*
`plugin:query-stage`	A *query stage plugin*

Examples

Upload a JavaScript file to the blob store:

REQUEST

```
curl -u USERNAME:PASSWORD -X PUT -H 'Content-Type: text/javascript' --data-binary
@query-pipeline-js-stage.js https://FUSION_HOST:FUSION_PORT/api/blobs/query-
pipeline-js-stage.js?resourceType=file:js-query
```

The response is empty.

Verify that the JavaScript blob was uploaded:

REQUEST

```
curl -u USERNAME:PASSWORD https://FUSION_HOST:FUSION_PORT/api/blobs?
resourceType=file:js-query
```

RESPONSE

```
[ { "id" : "query-pipeline-js-stage.js", "path" : "/query-pipeline-js-stage.js",
"dir" : "/", "filename" : "query-pipeline-js-stage.js", "contentType" :
"text/javascript", "size" : 0, "modifiedTime" : "2018-09-11T16:23:17.743Z",
"version" : 1611328911094841344, "md5" : "d41d8cd98f00b204e9800998ecf8427e",
"metadata" : { "resourceType" : "file:js-query" } } ]
```

Upload an OpenNLP sentence model binary file to the blob store:

REQUEST

```
curl -u USERNAME:PASSWORD -X PUT --data-binary @data/nlp/models/en-sent.bin -H
'Content-type: application/octet-stream'
https://FUSION_HOST:FUSION_PORT/api/blobs/sentenceModel.bin
```

	In this example, we have changed the name of the blob during upload by giving it a different ID. The file is named 'en-sent.bin' but we have defined the ID of this to 'sentenceModel.bin'. When we use this blob in an index pipeline, we would refer to it by the ID we have given it.

Get the manifest for a sentence OpenNLP model we have previously saved in the blob store:

REQUEST

```
curl -u USERNAME:PASSWORD -X HEAD
https://FUSION_HOST:FUSION_PORT/api/blobs/sentenceModel.bin
```

RESPONSE

```
{ "name" : "sentenceModel.bin", "contentType" : "application/octet-stream", "size"
: 98533, "modifiedTime" : "2014-09-08T18:50:07.559Z", "version" :
1478704189996531712, "md5" : "3822c5f82cb4ba139284631d2f6b7fde" }
```

Upload a JDBC driver, using slashes in the blob name:

REQUEST

```
curl -u USERNAME:PASSWORD -X PUT --data-binary @en-sent.bin -H 'Content-length: 6'
-H 'Content-type: application/octet-stream'
https://FUSION_HOST:FUSION_PORT/api/blobs/good/to/go/sentenceModel.bin?
resourceType=driver:jdbc
```

RESPONSE

```
{ "name" : "good/to/go/sentenceModel.bin", "contentType" : "application/octet-
stream", "size" : 6, "modifiedTime" : "2017-04-04T15:58:32.856Z", "version" : 0,
"md5" : "b1946ac92492d2347c6235b4d2611184", "metadata" : { "subtype" :
"driver:jdbc", "resourceType" : "driver:jdbc" } }
```

Get the JDBC driver that was uploaded:

REQUEST

```
curl -u USERNAME:PASSWORD https://FUSION_HOST:FUSION_PORT/api/blobs?
resourceType=driver:jdbc
```

RESPONSE

```
[ { "name" : "good/to/go/sentenceModel.bin", "contentType" : "application/octet-
stream", "size" : 6, "modifiedTime" : "2017-04-04T06:21:53.465Z", "version" :
1563727666574524416, "md5" : "b1946ac92492d2347c6235b4d2611184", "metadata" : {
"subtype" : "driver:jdbc", "resourceType" : "driver:jdbc" } } ]
```

Collection Features API

The Collection Features API lets you observe the status of features across collections. To manipulate features for a specific collection, use the *Collections API*.

Features

There are four available features:

Property	Description
signals	Create a parallel collection for the storage of signals data (such as user clicks, or ratings). Signals will need to be indexed and aggregated in order to be used. See the section on *Signals* for more information.
partitionByTime	Partition the corresponding Solr collection by time; see *Time-Based Partitioning*.
recommendations	Get configuration information about recommendations. Parameters (`params`) are: `idField`. The document field that contains the document ID. `itemsForUser`. Whether items-for-user recommendations are enabled (`true`) or disabled (`false`). `itemsForQuery`. Whether items-for-query recommendations are enabled (`true`) or disabled (`false`). `itemsForItem`. Whether items-for-query recommendations are enabled (`true`) or disabled (`false`). `queriesForQuery`. Whether queries-for-query recommendations are enabled (`true`) or disabled (`false`).

Examples

List the status of all features:

REQUEST

```
curl -u USERNAME:PASSWORD https://FUSION_HOST:FUSION_PORT/api/features/status
```

RESPONSE

```
[ { "name" : "searchLogs", "collectionId" : "demo", "params" : { }, "enabled" :
false }, { "name" : "signals", "collectionId" : "demo", "params" : { }, "enabled" :
false } ]
```

Collections API

The Collections API manages Fusion collections. It provides endpoints for creating, updating, and deleting collection, as well as endpoints for getting a collection's status and usage statistics.

Fusion maintains internal system collections for logs, blobs, and metrics data which operate in conjunction with collections created by users. The Collections API is used to manage all Fusion collections.

Examples

> In order to see this object within the *Fusion UI*, it **must** be associated with an app. To do this, create the object using the `/apps` endpoint.

Create a new collection called 'newCollection', with appropriate SolrCloud environment settings:

REQUEST

```
curl -u USERNAME:PASSWORD -X POST -H 'Content-type: application/json' -d
'{"solrParams":{"replicationFactor":1,"numShards":1}}'
https://FUSION_HOST:FUSION_PORT/api/collections/newCollection
```

RESPONSE

```
{ "id" : "newCollection", "createdAt" : "2014-09-19T18:46:52.954Z",
"searchClusterId" : "default", "solrParams" : { "name" : "newCollection",
"numShards" : 1, "replicationFactor" : 1 }, "type" : "DATA", "metadata" : { } }
```

Create a collection named 'local-collection1' that refers to 'collection1' in a pre-existing SolrCloud cluster named 'Solr4.10':

REQUEST

```
curl -u USERNAME:PASSWORD -X POST -H 'Content-type: application/json' -d
'{"id":"local-collection1", "searchClusterId":"Solr4.10", "solrParams":
{"name":"collection1"}}' https://FUSION_HOST:FUSION_PORT/api/collections
```

RESPONSE

```
{ "id" : "local-collection1", "createdAt" : "2014-09-19T18:48:45.396Z",
"searchClusterId" : "Solr4.10", "solrParams" : { "name" : "collection1" }, "type" :
"DATA", "metadata" : { } }
```

Delete a collection, but keep the associated signals and searchLogs collections:

REQUEST

```
curl -u USERNAME:PASSWORD -X DELETE
https://FUSION_HOST:FUSION_PORT/api/collections/newCollection?purge=false
```

Get the status of the 'demo' collection:

REQUEST

```
curl -u USERNAME:PASSWORD
https://FUSION_HOST:FUSION_PORT/api/collections/demo/status
```

RESPONSE

```
{ "maxShardsPerNode" : 1, "replicationFactor" : 1, "shards" : { "shard1" : {
"range" : "80000000-7fffffff", "state" : "active", "replicas" : { "core_node1" : {
"state" : "active", "core" : "demo_shard1_replica1", "leader" : true, "base_url" :
"https://FUSION_HOST:8983/solr", "node_name" : "FUSION_HOST:8983_solr" } } } } }
```

Get stats for the 'demo' collection:

REQUEST

```
curl -u USERNAME:PASSWORD
https://FUSION_HOST:FUSION_PORT/api/collections/demo/stats
```

RESPONSE

```
{ "collectionId" : "demo", "documentCount" : 536, "requestCount" : 6, "qps" :
28.34716542561849, "sizeInBytes" : 7646045, "lastModified" : "2014-05-
19T19:58:33.545Z" }
```

Add the recommendations feature for the 'tempy' collection:

For more information on collection features and feature management, see *Collection Features API*.

Use this structure to send the request:

```
https://FUSION_HOST:FUSION_PORT/api/collections/{collection}/features/{feature}
```

```
curl -u USERNAME:PASSWORD -X POST -H 'Content-type: application/json' -d @enable-recommendations.json https://FUSION_HOST:FUSION_PORT/api/apps/Recommender-App/collections/tempy/features/recommendations -v
```

Make sure to change the `collection-id` in the JSON and save as a file or send directly in the string.

Recommendations JSON:

```
{ "name": "recommendations", "collectionId": "tempy", "params": { "idField": "id", "itemsForUser": true, "itemsForQuery": false, "itemsForItem": true, "queriesForQuery": false } , "enabled": true }
```

Signals JSON:

```
{ "name": "signals", "collectionId": "tempy", "params": {}, "enabled": false },
```

Search Logs JSON:

```
{ "name": "searchLogs", "collectionId": "tempy", "params": {}, "enabled": false }
```

Partition by time JSON:

```
{ "name": "partitionByTime", "collectionId": "tempy", "params": {}, "enabled": false }
```

Configurations API

The Configurations API allows setting global properties for Fusion.

Some settings are not set by any configuration file but are reported as settings from the operating system. Those settings cannot be changed with this API.

Examples

Show the configuration items that include the pattern 'zk-connect', with verbose enabled:

REQUEST

```
curl -u USERNAME:PASSWORD https://FUSION_HOST:FUSION_PORT/api/configurations?
verbose=true&pattern=zk-connect
```

RESPONSE

```
{
  "com.lucidworks.apollo-admin.config/zk-connect" : [ {
    "value" : "FUSION_HOST:9983",
    "location" : "system properties"
  } ]
}
```

Get the configuration items from the Connectors JVM that include the term 'connect':

REQUEST

```
curl -u USERNAME:PASSWORD
https://FUSION_HOST:8984/connectors/api/v1/configurations?pattern=connect
```

Get the configuration items from the Fusion JVM that start with 'com.lucidworks':

REQUEST

```
https://FUSION_HOST:FUSION_PORT/api/configurations?prefix=com.lucidworks
```

Change the default allowed recommendation types to include 'itemsForItem":

REQUEST

```
curl -u USERNAME:PASSWORD -X PUT -H 'Content-type: application/json' -d
'{"itemsForQuery", "termsForDocument", "itemsforItem"}'
https://FUSION_HOST:FUSION_PORT/api/configurations/com.lucidworks.apollo.service.re
commend.allowed.types
```

RESPONSE

None. Check the setting again to confirm the changes.

Data Models API

Manages data models.

Data models simplify the process of getting started with Fusion by providing pre-configured objects to reduce the effort spent on basic starting tasks. This helps keep documents consistent between datasources and intuitive to the object's type.

UI fields configuration

Field	Required	Description	Example
Name of the Data Model	✅	`string` A user-friendly name for the data model. This typically describes the object type.	`person`
Index Pipeline	✅	`string` The name of the index pipeline used for the data model.	`companyDirectory-index-pipeline`
Query Pipeline	✅	`string` The name of the query pipeline used for the data model.	`companyDirectory-query-pipeline`
Field Name	✅	`string`	`first_name`
Required	✘	`boolean`	`true`
Solr Fields ↓			
Solr Field Name	✅	`string`	`firstName_t`
Is Query Field	✘	`boolean`	`true`
Enable Phrase Match	✘	`boolean`	`false`
Boost Value	✘	`integer` The amount of boost to give to the query.	`1`
Phrase Boost	✘	`integer` The amount of boost to give to the query, if it matches as an exact phrase.	`2`

> The value you put in **Name of the Data Model** is automatically converted to lowercase for the `id` field in the backend. Use lowercase syntax for API calls. For a list of data models and their `id` values, visit `https://FUSION_HOST:FUSION_PORT/api/data-models`.

More information

- *How to Configure a Data Model*
- *Data Models*
- *Data Model Mapping Index Stage reference*
- *Call Data Model Pipeline Index Stage reference*

History API

Fusion stores history for each running service within the system. Usually this is used to log start and stop events for a service. However, the scheduler uses the history to store the results of scheduled tasks. For more information on schedule history, see the section on *Jobs and Schedules*.

The History API provides information about the services that are running. The list of these services is provided by Introspect API, which is described in the *REST API Reference*.

Examples

View the history of the index-pipelines service:

REQUEST

```
curl -u USERNAME:PASSWORD https://FUSION_HOST:FUSION_PORT/api/history/index-pipelines::v1
```

RESPONSE

```
{
  "events" : [ {
    "start" : "2014-05-16T14:11:48.849Z",
    "end" : "2014-05-16T14:11:48.849Z",
    "source" : "index-pipelines::v1",
    "type" : "start",
    "status" : "ok",
    "details" : null,
    "error" : null
  }, {
    "start" : "2014-05-16T14:12:48.845Z",
    "end" : "2014-05-16T14:12:48.845Z",
    "source" : "index-pipelines::v1",
    "type" : "start",
    "status" : "ok",
    "details" : null,
    "error" : null
  }
}
```

View items in the scheduler history:

REQUEST

```
curl -u USERNAME:PASSWORD
https://FUSION_HOST:FUSION_PORT/api/history/scheduler/items/1
```

RESPONSE

```
{
  "events" : [ {
    "start" : "2014-05-16T15:34:49.008Z",
    "end" : "2014-05-16T15:34:49.435Z",
    "source" : "scheduler",
    "type" : "execute",
    "status" : "ok",
    "details" : {
      "status" : 200,
      "entity" : "{\n  \"id\" : \"TwitterSearch\",\n  \"dataSourceId\" :
\"TwitterSearch\",\n  \"state\" : \"RUNNING\",\n  \"message\" : null,\n
\"startTime\" : 1400254489000,\n  \"endTime\" : -1,\n  \"finished\" : false,\n
\"counters\" : { },\n  \"exception\" : null,\n  \"running\" : true\n}"
    },
    "error" : null
  }, {
    "start" : "2014-05-16T15:38:32.536Z",
    "end" : "2014-05-16T15:38:32.559Z",
    "source" : "scheduler",
    "type" : "execute",
    "status" : "ok",
    "details" : {
      "status" : 200,
      "entity" : "{\n  \"id\" : \"TwitterSearch\",\n  \"dataSourceId\" :
\"TwitterSearch\",\n  \"state\" : \"RUNNING\",\n  \"message\" : null,\n
\"startTime\" : 1400254712000,\n  \"endTime\" : -1,\n  \"finished\" : false,\n
\"counters\" : { },\n  \"exception\" : null,\n  \"running\" : true\n}"
    },
    "error" : null
  }
}
```

Jobs API

The Jobs API allows you to define schedules for objects that perform work. You can schedule datasources, tasks, and Spark jobs.

This topic explains how to format requests to the Jobs API. To learn how to work with jobs and schedules in the Fusion UI, see *Jobs and Schedules*.

Job runtime configuration

	In order to see this object within the *Fusion UI*, it **must** be associated with an app. To do this, create the object using the `/apps` endpoint.

Job configuration requests have these parameters:

- `<type>:<id>` path parameter (required)
  The resource identifier takes the form `<type>:<id>`, such as `datasource:movie-db`, `spark:dailyMetricsRollup-counters`, or `task:delete-old-system-logs`.
  See *Job types* below.
- `enabled` body parameter
  "True" or "false" to enable or disable the job.
- `triggers` body parameter
  This parameter defines one or more conditions that trigger the job. See *Job triggers* below.

Below is an example request that configures a job:

```
PUT /jobs/datasource:movie-db/schedule { "enabled": true, "triggers": [ {"type":
"cron", "enabled": true, "value": "0 */10 * * * *"} ] }
```

Job types

The job type must be specified as part of the resource identifier in `PUT`, `POST`, and `DELETE` calls. These are the possible job types:

`datasource`	A job to ingest data according to the specified datasource configuration, such as `datasource:movie-db`. Datasources are created using the *Connector Datasources API* or the Fusion UI. See *Datasource Jobs*.
`spark`	A Spark job to process data, such as `spark:dailyMetricsRollup-counters`. Spark jobs are created using the *Spark Jobs API* or the Fusion UI. See *Spark Jobs*.
`task`	A job to perform an HTTP call or log cleanup, such as `task:delete-old-system-logs`. Tasks are created using the Fusion UI. See *Tasks*.

Job triggers

Each job can have multiple triggers, and each trigger configuration requires `enabled` and `type` parameters.

Additional parameters are required by each trigger type, described below:

Trigger type	Parameters
`interval` Run the job at a regular interval over some time range.	• `startTime` The date and time at which the job will first run, as an ISO-8601 date-time string *(https://www.w3.org/TR/NOTE-datetime)*. • `endTime` The date and time at which the job will run last, as an ISO-8601 date-time string. • `interval` The interval at which the job will run, as an integer whose unit is specified by `repeatUnit` below. • `repeatUnit` One of: `minute`, `hour`, or `day`.
`cron` Run the job on a cron-style schedule.	`value` A cron string.
`job` Run the job after another job succeeds or fails.	• `jobID` The ID of the job that triggers this one. • `runOnSuccess` "True" to run when the specified job succeeds. • `runOnFailure` "True" to run when the specified job fails. • `runOnAbort` "True" to run when the specified job is aborted.

Examples

Get all datasource jobs

Input

```
curl -u USERNAME:PASSWORD https://FUSION_HOST:FUSION_PORT/api/jobs?type=datasource
```

Output

```
[ { "resource" : "datasource:ratings-db", "enabled" : true, "startedBy" :
"acb38917-611a-46e1-a3cf-4ed46a30fd24" }, { "resource" : "datasource:movies-db",
"enabled" : true, "startedBy" : "acb38917-611a-46e1-a3cf-4ed46a30fd24", "status" :
"success", "lastStartTime" : "2017-05-17T20:58:10.503Z", "lastEndTime" : "2017-05-
17T20:58:42.020Z" } ]
```

Start a datasource job now

Input

```
curl -u USERNAME:PASSWORD -X POST -H "Content-Type: application/json"
https://FUSION_HOST:FUSION_PORT/api/jobs/datasource:movies-db/actions -d
'{"action": "start"}'
```

Output

```
{ "resource" : "datasource:movies-db", "action" : "start", "accepted" : true,
"message" : "Started job datasource:movies-db" }
```

Get the job history for a datasource

Input

```
curl -u USERNAME:PASSWORD
https://FUSION_HOST:FUSION_PORT/api/jobs/datasource:movies-db/history
```

Output

```
[ { "resource" : "datasource:movies-db", "startTime" : "2017-05-17T21:01:46.743Z",
"endTime" : "2017-05-17T21:02:05.720Z", "status" : "success" }, { "resource" :
"datasource:movies-db", "startTime" : "2017-05-17T20:58:10.503Z", "endTime" :
"2017-05-17T20:58:42.020Z", "status" : "success" } ]
```

Define a trigger for a datasource job

Input

```
curl -u USERNAME:PASSWORD -X PUT -H "Content-Type: application/json"
https://FUSION_HOST:FUSION_PORT/api/jobs/datasource:ratings-db/schedule -d
'{"triggers": [{"type": "cron", "enabled": "true", "expression": "0 15 10 ? * MON-
FRI"}]}'
```

Output

```
{ "resource" : "datasource:ratings-db", "triggers" : [ { "type" : "cron", "enabled"
: true, "expression" : "0 15 10 ? * MON-FRI", "type" : "cron" } ] }
```

Links API

The Links API manages the links that represent the relationships between Fusion objects. You can see these links in the *Object Explorer* by pressing CMD-K or CTRL-K from any screen in the Fusion UI. You can also export and import links using the *Objects API*.

Links are structured as a tuple of `(subject, object, linkType)`. An example would be `(foo, bar, dependsOn)`, which would read as "foo dependsOn bar".

Link types and link reversibility

The possible values of `linkType` are as follows:

`inContextOf`	The subject exists in the context of the object, which is an app. In this example, the "Movie_Search" app is the context for a parser with the same name: `{ "subject" : "parser:Movie_Search", "object" : "app:Movie_Search", "linkType" : "inContextOf", "originator" : "unspecified" }`
`hasContext`	The subject is an app that contains the object. In this example, an index profile exists in the context of the "Movie_Search" app: `{ "subject" : "app:Movie_Search", "object" : "index-profile:Movie_Search", "linkType" : "hasContext", "originator" : "unspecified" }`
`dependsOn`	The subject depends on the object. In this example, a datasource called "web_pokeapi_co-default" depends on a parser by the same name: `{ "subject" : "datasource:web_pokeapi_co-default", "object" : "parser:web_pokeapi_co-default", "linkType" : "dependsOn" }`
`supports`	The subject is depended on by the object (the reverse of `dependsOn`). In this example, a parser called "web_pokeapi_co-default" supports a datasource by the same name: `{ "subject" : "parser:web_pokeapi_co-default", "object" : "datasource:web_pokeapi_co-default", "linkType" : "supports" }`
`isPartOf`	The subject is part of the object, where the object is a group. In this example, an aggregation called "click-signals-default" is part of a group called "signals-object-group-default": `{ "subject" : "aggregation:click-signals-default", "object" : "group:signals-object-group-default", "linkType" : "isPartOf" }`

`hasPart`	The subject is a group, of which the object is a member (the reverse of `isPartOf`). In this example, an aggregation called "click-signals-default" is part of a group called "signals-object-group-default": `{ "subject" : "group:signals-object-group-default", "object" : "aggregation:click-signals-default", "linkType" : "hasPart" }`
`relatesTo`	The subject relates to the object. This is a uni-directional relationship that associates two objects that have none of the relationships described above. For example, when recommendations are enabled for the default collection, Fusion creates a group of objects which relates to the default collection: `{ "subject" : "group:recommendations-default", "object" : "collection:default", "linkType" : "relatesTo" }`

With the exception of the `relatesTo` link type, links are reversible:

- `inContextOf` is the reverse of `hasContext` (and vice versa).
- `dependsOn` is the reverse of `supports` (and vice versa).
- `isPartOf` is the reverse of `hasPart` (and vice versa).

For example, here is the group of links for the `signals-default` collection:

```
[ { "subject": "group:4e6ee9a6-d4ab-413a-ac70-a27c1445bb60", "object": "collection:default_signals_aggr", "linkType": "hasPart" }, { "subject": "group:4e6ee9a6-d4ab-413a-ac70-a27c1445bb60", "object": "spark:default_click_signals_aggregation", "linkType": "hasPart" }, { "subject": "group:4e6ee9a6-d4ab-413a-ac70-a27c1445bb60", "object": "collection:default", "linkType": "relatesTo" }, { "subject": "group:4e6ee9a6-d4ab-413a-ac70-a27c1445bb60", "object": "collection:default_signals", "linkType": "hasPart" } ]
```

When we request the links for one of the `hasPart` objects shown above, we get the reverse representation:

```
GET https://FUSION_HOST:FUSION_PORT/api/links?subject=collection:default_signals_aggr [ { "subject": "collection:default_signals_aggr", "object": "group:4e6ee9a6-d4ab-413a-ac70-a27c1445bb60", "linkType": "isPartOf" } ]
```

Examples

Get links to the default collection

```
curl https://FUSION_HOST:FUSION_PORT/api/links?subject=collection:default
```

Output:

```
[ { "subject": "collection:default", "object": "index-pipeline:default",
"linkType": "supports" } ]
```

Create a new link

```
curl -X PUT https://FUSION_HOST:FUSION_PORT/api/links -H 'content-type:
application/json' -d '{"linkType": "supports", "object": "index-pipeline:default",
"subject": "collection:default"}'
```

Output:

```
{ "subject" : "collection:default", "object" : "index-pipeline:default", "linkType"
: "supports" }
```

Delete a link

```
curl -X DELETE https://FUSION_HOST:FUSION_PORT/api/links?
subject=collection:default
```

Nodes API

The Nodes API lets you check the status of each node in the search cluster.

Examples

Show each service:

REQUEST

```
curl –u USERNAME:PASSWORD https://FUSION_HOST:FUSION_PORT/api/nodes/up
```

RESPONSE

```
{ "history::v1" : [ "https://FUSION_HOST:8984/connectors/v1/history",
"https://FUSION_HOST:FUSION_PORT/api/history" ], "query-pipelines::v1" : [
"https://FUSION_HOST:FUSION_PORT/api/query-pipelines",
"https://FUSION_HOST:FUSION_PORT/api/query-pipelines" ], "solrconfig::v1" : [
"https://FUSION_HOST:FUSION_PORT/api/collections//solr-config" ], "solrAdmin::v1" :
[ "https://FUSION_HOST:FUSION_PORT/api/solrAdmin",
"https://FUSION_HOST:8984/connectors/v1/solrAdmin" ], "stopwords::v1" : [
"https://FUSION_HOST:FUSION_PORT/api/stopwords" ], "collections::v1" : [
"https://FUSION_HOST:FUSION_PORT/api/collections" ], "index-stages::v1" : [
"https://FUSION_HOST:8984/connectors/v1/index-stages",
"https://FUSION_HOST:FUSION_PORT/api/index-stages" ], "nodes::v1" : [
"https://FUSION_HOST:8984/connectors/v1/nodes",
"https://FUSION_HOST:FUSION_PORT/api/nodes" ], "index-profiles::v1" : [
"https://FUSION_HOST:FUSION_PORT/api/collections//index-profiles",
"https://FUSION_HOST:8984/connectors/v1/collections//index-profiles" ],
"schema::v1" : [ "https://FUSION_HOST:FUSION_PORT/api/collections//schema" ],
"solr::v1" : [ "https://FUSION_HOST:8984/connectors/v1/solr",
"https://FUSION_HOST:FUSION_PORT/api/solr" ], "query-stages::v1" : [
"https://FUSION_HOST:FUSION_PORT/api/query-stages" ], "blobs::v1" : [
"https://FUSION_HOST:FUSION_PORT/api/blobs" ], "connectors::v1" : [
"https://FUSION_HOST:8984/connectors/v1/connectors" ], "recommend::v1" : [
"https://FUSION_HOST:FUSION_PORT/api/recommend" ], "configurations::v1" : [
"https://FUSION_HOST:FUSION_PORT/api/configurations",
"https://FUSION_HOST:8984/connectors/v1/configurations" ], "system::v1" : [
"https://FUSION_HOST:FUSION_PORT/api/system",
"https://FUSION_HOST:8984/connectors/v1/system" ], "index-pipelines::v1" : [
"https://FUSION_HOST:FUSION_PORT/api/index-pipelines",
"https://FUSION_HOST:FUSION_PORT/api/index-pipelines",
"https://FUSION_HOST:FUSION_PORT/api/index-pipelines",
"https://FUSION_HOST:FUSION_PORT/api/index-pipelines",
"https://FUSION_HOST:FUSION_PORT/api/index-pipelines",
"https://FUSION_HOST:FUSION_PORT/api/index-pipelines",
"https://FUSION_HOST:FUSION_PORT/api/index-pipelines",
"https://FUSION_HOST:8984/connectors/v1/index-pipelines",
"https://FUSION_HOST:8984/connectors/v1/index-pipelines",
"https://FUSION_HOST:FUSION_PORT/api/index-pipelines",
"https://FUSION_HOST:FUSION_PORT/api/index-pipelines" ], "query-pipeline-
templates::v1" : [ "https://FUSION_HOST:FUSION_PORT/api/pipeline-templates/query"
], "registration::v1" : [ "https://FUSION_HOST:8984/connectors/v1/registration",
"https://FUSION_HOST:FUSION_PORT/api/registration" ], "searchLogs::v1" : [
"https://FUSION_HOST:FUSION_PORT/api/searchLogs" ], "scheduler::v1" : [
"https://FUSION_HOST:FUSION_PORT/api/scheduler" ], "aggregator::v1" : [
"https://FUSION_HOST:FUSION_PORT/api/aggregator" ], "query-profiles::v1" : [
"https://FUSION_HOST:FUSION_PORT/api/collections//query-profiles" ], "usage::v1" :
[ "https://FUSION_HOST:FUSION_PORT/api/usage",
"https://FUSION_HOST:8984/connectors/v1/usage" ], "introspect::v1" : [
"https://FUSION_HOST:FUSION_PORT/api/introspect",
"https://FUSION_HOST:8984/connectors/v1/introspect" ], "searchCluster::v1" : [
"https://FUSION_HOST:FUSION_PORT/api/searchCluster",
"https://FUSION_HOST:8984/connectors/v1/searchCluster" ], "features::v1" : [
"https://FUSION_HOST:FUSION_PORT/api/features" ], "index-pipeline-templates::v1" :
[ "https://FUSION_HOST:8984/connectors/v1/pipeline-templates/index",
"https://FUSION_HOST:FUSION_PORT/api/pipeline-templates/index" ], "synonyms::v1" :
[ "https://FUSION_HOST:FUSION_PORT/api/synonyms" ], "reports::v1" : [
"https://FUSION_HOST:FUSION_PORT/api/reports" ], "signals::v1" : [
"https://FUSION_HOST:FUSION_PORT/api/signals" ] }
```

Objects API

The Objects API lets you import and export objects between Fusion instances. Fusion objects include all your searchable data, plus pipelines, aggregations, and other configurations on which your collections depend.

As an alternative in some cases, you can *use the Fusion UI to migrate objects*.

	For both the API and UI, migrating objects is intended for use *within a specific Fusion version*. See *Fusion 5 Upgrades* to migrate objects from one Fusion version to a specific Fusion 5.x release.

You can select all objects, or limit the operation to specific object types or IDs. In addition to export/import endpoints, a validation endpoint is provided for troubleshooting.

	By default, system-created collections are not exported. See below for details.

Object export and import

Collections and *encrypted values* are treated specially; details are provided below. During import, conflicts are resolved according to the specified *import policy*.

For objects other than collections, no implicit filtering is performed; all objects are included by default. However, on export you can *filter by type and ID*.

Supported objects

Fusion lets you export and import object types. ZIP files are the output for all file types listed. To filter by type, use the values below with the `type` query string parameter; see the examples below.

Object type	Value
apps	`app` The `deep` parameter is not needed for this object type. In the case of apps, all linked objects are included by default.
Appkit apps	`appkit`
blob	`blob`
collections (with their dependent objects) See below for collection-specific details.	`collection`
data-model	`data-model`
datasources	`datasource`
experiment	`experiment`
index pipelines	`index-pipeline`
index profiles	`index-profile`
job configurations	`spark`, `type`, or `data-source`
link	`link`
object group	`group`
parsing configurations	`parser`
query pipelines	`query-pipeline`
query profiles	`query-profile`
schedules for schedulable jobs (spark, type, or datasource)	`job`
search cluster configurations	`search-cluster`
subscription	`subscription`
template	`template`
zone	`zone`

Exporting and importing collections

Collections are processed with these dependent objects:

- features

- index profiles
- query profiles

Datasources, parser configurations, and pipeline configurations are not included when collections are exported or imported. These must be exported and imported explicitly.

Only user-created collections are included by default. Certain types of collections are excluded:

- the "default" collection
- collections whose type is not DATA
- collections whose names start with "system_"
- "Secondary" collections, that is, collections created by features
  Instead, create the same features on the target system; this automatically creates the corresponding secondary collections.

You can override these exclusions by specifying a collection, like this:

```
curl -u USERNAME:PASSWORD https://FUSION_HOST:FUSION_PORT/api/objects/export?
collection.ids=default > collection.zip
```

Encrypted passwords

Some objects, such as datasources and query pipelines, include encrypted passwords for accessing secure data.

- On *export*, these encrypted values are replaced with one of the following:
  - `secret.dataSources.datasourceID.password`
  - `secret.queryPipelines.pipelineID.stages.active-directory-security-trimming.stageID.bindPassword`
- On *import*, the original, plaintext passwords must be provided in a separate JSON map as multipart form data:

```
{
   "secret.dataSources.datasourceID.password":"MyDatasourcePassword",
   "secret.queryPipelines.pipelineID.stages.active-directory-security-
trimming.stageID.bindPassword":"MyADPassword"
}
```

> Passwords can also be *uploaded via the UI*. Variables that do not start with `secret.` are ignored.

Import policies

On import, the `importPolicy` parameter is required. It specifies what to do if any object in the import list already exists on the target system:

`abort`	If there are conflicts, then import nothing.
`merge`	If there are conflicts, then skip the conflicting objects.
`overwrite`	If there are conflicts, then overwrite or delete/create the conflicting objects on the target system.

Filtering on export

On export, there are two ways to specify the objects to include:

- by type
  To export only specific types of objects, you can specify one or more object types using the `type` query string parameter. See *Supported objects* for a list of valid values.

- by type and ID
  The `type.ids` parameter lets you list the IDs to match for the specified object type.

The `type` and `type.ids` parameters can be combined as needed.

Exporting linked objects

Related Fusion objects are linked. You can view linked objects using the *Links API* or the *Object Explorer*.

When exporting a specific Fusion object, you can also export its linked objects without specifying each one individually. To export all objects linked to the specified object, include the `deep="true"` query parameter in your request. See the *example* below.

Validation

Objects are validated before import. If any objects fail validation, the whole import request is rejected. A separate endpoint is available for validating objects without importing them.

Validation includes checking whether an object already exists on the target system and whether the user is authorized to create or modify the object.

For collection objects, the following special validation is performed:

- We check the `searchClusterId` of each collection and verify that a cluster with this ID exists on the target system or in the import file (error).
- We check that features, index profiles, and query profiles belong only to the collections specified in the import file (error).
- We check that a feature exists on the target system for each feature in the import file (error).
- We check for index profiles or query profiles that do not exist on the target system or in the import file (warning).

Status messages

`Validation completed with no errors`	The validation method was called and no errors found, though there may be warnings.
`Validation found errors`	The validation was called and errors found. Validation does not stop on the first error, so the complete list of errors is reported.
`Validation was not completed because of system error`	The validation was interrupted by system error.
`Import was not performed because validation errors exist`	The import method was called, but import didn't start because of validation errors.
`Import was not performed because of input data error`	The import method was called, but import didn't start, because Fusion could not find a substitution for one of the secret values in objects in import.
`Import was not completed because of system error`	The validation found no errors and import started, but it was interrupted by system error.
`Import was completed`	Validation found no errors and import finished successfully.

Examples

Export all objects

```
curl -u USERNAME:PASSWORD https://FUSION_HOST:FUSION_PORT/api/objects/export > all-objects.zip
```

Export all datasources

```
curl -u USERNAME:PASSWORD https://FUSION_HOST:FUSION_PORT/api/objects/export?
type=datasource > datasources.zip
```

Export a specific datasource and all its linked objects

```
curl -u USERNAME:PASSWORD https://FUSION_HOST:FUSION_PORT/api/objects/export?
datasource.ids=movies_csv-movies&deep=true > movies.zip
```

Export all apps

```
curl -u USERNAME:PASSWORD https://FUSION_HOST:FUSION_PORT/api/objects/export?
type=app > all-apps.zip
```

Get all app IDs, then export one app by ID

```
curl -u USERNAME:PASSWORD https://FUSION_HOST:FUSION_PORT/api/apps [ { "id" :
"movies", "name" : "Movies", "description" : "Search the movielens database.",
"dataUri" : "/App-Tile-01-460x160.png", "properties" : { "headerImageName" :
"headerImage1", "tileColor" : "apps-darkblue", "previousCollectionId" : "movies" }
}, { "id" : "tech-pubs", "name" : "TechPubs", "description" : "Search the
documentation.", "dataUri" : "/App-Tile-02-460x160.png", "properties" : {
"headerImageName" : "headerImage2", "tileColor" : "apps-darkblue",
"previousCollectionId" : "tech-pubs" } } ] curl -u USERNAME:PASSWORD
https://FUSION_HOST:FUSION_PORT/api/objects/export?app.ids=tech-pubs > techpubs.zip
```

When you export an app that includes objects that are shared with other apps, then all apps linked to that object are also exported.

Export two apps by ID

```
curl -u USERNAME:PASSWORD https://FUSION_HOST:FUSION_PORT/api/objects/export?
app.ids=tech-pubs,movies > two-apps.zip
```

Export all datasources and pipelines, plus two specific parsing configurations

```
curl -u USERNAME:PASSWORD 'https://FUSION_HOST:FUSION_PORT/api/objects/export?
type=datasource,index-pipeline,query-pipeline&parser.ids=movies,tech-pubs' >
export.zip
```

Import objects from a file and stop if there are conflicts

```
curl -u USERNAME:PASSWORD -H "Content-Type:multipart/form-data" -X POST -F
'importData=@/Users/admin/Fusion/export.zip'
https://FUSION_HOST:FUSION_PORT/api/objects/import?importPolicy=abort
```

Import objects, substitute the password variables, and merge any conflicts

```
curl -u USERNAME:PASSWORD -H "Content-Type:multipart/form-data" -X POST -F
'importData=@/Users/admin/Fusion/export.zip' -F
'variableValues=@password_file.json'
https://FUSION_HOST:FUSION_PORT/api/objects/import?importPolicy=merge
```

`password_file.json` must contain plaintext passwords.

Import a zip file of Fusion objects and merge any conflicts

```
curl -u USERNAME:PASSWORD -H "Content-Type:multipart/form-data" -X POST -F
'importData=@/Users/admin/Fusion/techpubs.zip'
https://FUSION_HOST:FUSION_PORT/api/objects/import?importPolicy=merge
```

Import objects into the context of an app

```
curl -v -H 'Content-Type:multipart/form-data' -X POST -F 'importData=@app_part.zip'
'https://FUSION_HOST:8765/api/v1/objects/import?
importPolicy=overwrite&context=APP_NAME'
```

Validate objects

```
curl -v -H 'Content-Type:multipart/form-data' -X POST -F
'https://FUSION_HOST:FUSION_PORT/api/v1/objects/validate'
```

Scheduler API

Use the Scheduler API to define a schedule for system activities and manage the jobs that result from the schedule.

This API is deprecated in favor of the *Jobs API*.

All of the Fusion services are available for scheduling, as are any Solr activities and any other HTTP-based URI.

See *Jobs and Schedules* for more information about all of the options available for scheduling, and instructions for configuring scheduler jobs using the Fusion UI.

Schedule Definition Properties

Parameter	Description
id *Required*	The ID of the schedule. This is required when creating a schedule with a POST request.
creatorType *Optional*	The type of user that created the schedule. If you have resources creating schedules programmatically, you could define a type that helps distinguish those schedules from others created by people.
creatorId *Optional*	An ID for the user that created the schedule.
startTime *Optional*	A time when the schedule should be started. If this is not set, it will be set to the date/time when the schedule was created.
endTime *Optional*	A time when the schedule should be terminated, i.e., when it will not run any more in the future. If not set, the schedule will run until it is disabled (i.e., "active" is set to false) or deleted.
repeatUnit *Optional*	A unit of time that when combined with the interval property defines how often the schedule will run. Allowed values are: • "millisecond" or "ms" • "second" or "sec" • "minute" or "min" • "hour" or "hr" • "day" • "week" • "month" These time units are not case sensitive.
interval *Optional*	An integer that when combined with the repeatUnit property defines how often the schedule will run. If this is not set, or set to a number lower than '1' (i.e., '0'), the schedule will only be run once.
active *Optional*	If **true**, the schedule will be executed according to the defined interval. If **false**, the schedule will be disabled and will not run at the defined time.

Parameter	Description

callParams *Required*	The callParams define the API call to execute at the defined time intervals. It allows several properties:
	• **uri.** a fully-qualified service URI. This can be an HTTP call, a Solr request, or a Fusion service call. Supported URI schemas are:
	  ◦ `http://` or `https://`
	  ◦ `solr://{collection}/…`
	  A SolrCloud request.
	  ◦ `service://{serviceName}/{path}`
	  A load-balanced Fusion service request.
	• **method.** The method to use, as in GET, POST, PUT, or DELETE.
	• **queryParams.** Query parameters to be passed with the request. For Solr calls, this could be parameters such as "q", "fq", "commit", etc.
	• **headers.** Any additional protocol headers, such as "Content-type".
	• **entity.** An optional payload to be sent with the request.

Examples

Run a crawl of the datasource "LocalDocs2" every day:

REQUEST

```
curl -u USERNAME:PASSWORD -X POST -H 'Content-Type: application/json' -d
'{"id":"1", "creatorType":"human", "creatorId":"admin1", "repeatUnit":"DAY",
"interval":1, "active":true, "callParams":
{"uri":"service://connectors/jobs/LocalDocs2", "method":"POST"}}'
https://FUSION_HOST:FUSION_PORT/api/scheduler/schedules
```

RESPONSE

```
{
  "id" : "1",
  "trigger" : "1_1400007488512_-9223372036854775808",
  "schedule" : {
    "id" : "1",
    "creatorType" : "human",
    "creatorId" : "admin1",
    "createTime" : "2014-05-13T18:58:08.512Z",
    "startTime" : "2014-05-13T18:58:08.512Z",
    "endTime" : null,
    "repeatUnit" : "DAY",
    "interval" : 1,
    "active" : true,
    "callParams" : {
      "uri" : "service://connectors/jobs/LocalDocs2",
      "method" : "POST",
      "queryParams" : { },
      "headers" : { },
      "entity" : null
    }
  }
}
```

Issue a commit to the SolrCloud collection "demo" every hour:

REQUEST

```
curl-u USERNAME:PASSWORD -X POST -H 'Content-Type: application/json' -d '{"id":"2",
"creatorType":"human", "creatorId":"admin1", "repeatUnit":"HOUR", "interval":1,
"active":true, "callParams":{"uri":"solr://demo/update", "method":"GET",
"queryParams":{"stream.body":"<commit/>"}}}'
https://FUSION_HOST:FUSION_PORT/api/scheduler/schedules
```

RESPONSE

```
{
  "id" : "2",
  "trigger" : "2_1400011854443_-9223372036854775807",
  "schedule" : {
    "id" : "2",
    "creatorType" : "human",
    "creatorId" : "admin1",
    "createTime" : "2014-05-13T20:10:54.443Z",
    "startTime" : "2014-05-13T20:10:54.443Z",
    "endTime" : null,
    "repeatUnit" : "HOUR",
    "interval" : 1,
    "active" : true,
    "callParams" : {
      "uri" : "solr://demo/update",
      "method" : "GET",
      "queryParams" : {
        "stream.body" : "<commit/>"
      },
      "headers" : { },
      "entity" : null
    }
  }
}
```

Set schedule "2" to inactive:

REQUEST

```
curl -u USERNAME:PASSWORD -X PUT -H 'Content-Type: application/json' -d
'{"creatorType":"human", "creatorId":"admin1", "repeatUnit":"HOUR", "interval":1,
"active":false, "callParams":{"uri":"solr://demo/update", "method":"GET",
"queryParams":{"stream.body":"<commit/>"}}}'
https://FUSION_HOST:FUSION_PORT/api/scheduler/schedules/2
```

There will be no response if the PUT request was successful.

List all scheduled jobs:

REQUEST

```
curl -u USERNAME:PASSWORD https://FUSION_HOST:FUSION_PORT/api/scheduler/jobs
```

RESPONSE

```
[ {
  "id" : "2",
  "trigger" : "2_1400011854443_-9223372036854775807",
  "schedule" : {
    "id" : "2",
    "creatorType" : "human",
    "creatorId" : "admin1",
    "createTime" : "2014-05-13T20:10:54.443Z",
    "startTime" : "2014-05-13T20:10:54.443Z",
    "endTime" : null,
    "repeatUnit" : "HOUR",
    "interval" : 1,
    "active" : true,
    "callParams" : {
      "uri" : "solr://demo/update",
      "method" : "GET",
      "queryParams" : {
        "stream.body" : "<commit/>"
      },
      "headers" : { },
      "entity" : null
    }
  }
}, {
  "id" : "1",
  "trigger" : "1_1400007488512_-9223372036854775808",
  "schedule" : {
    "id" : "1",
    "creatorType" : "human",
    "creatorId" : "admin1",
    "createTime" : "2014-05-13T18:58:08.512Z",
    "startTime" : "2014-05-13T18:58:08.512Z",
    "endTime" : null,
    "repeatUnit" : "DAY",
    "interval" : 1,
    "active" : true,
    "callParams" : {
      "uri" : "service://connectors/jobs/LocalDocs2",
      "method" : "POST",
      "queryParams" : { },
      "headers" : { },
      "entity" : null
    }
  }
} ]
```

List the history of job ID '1':

REQUEST

```
curl -u USERNAME:PASSWORD
https://FUSION_HOST:FUSION_PORT/api/history/scheduler/items/1
```

RESPONSE

```
{
 "events" : [ {
 "start" : "2014-05-16T15:34:49.008Z",
 "end" : "2014-05-16T15:34:49.435Z",
 "source" : "scheduler",
 "type" : "execute",
 "status" : "ok",
 "details" : {
 "status" : 200,
 "entity" : "{\n \"id\" : \"TwitterSearch\",\n \"dataSourceId\" :
\"TwitterSearch\",\n \"state\" : \"RUNNING\",\n \"message\" : null,\n \"startTime\"
: 1400254489000,\n \"endTime\" : -1,\n \"finished\" : false,\n \"counters\" : {
},\n \"exception\" : null,\n \"running\" : true\n}"
 },
 "error" : null
 }, {
 "start" : "2014-05-16T15:38:32.536Z",
 "end" : "2014-05-16T15:38:32.559Z",
 "source" : "scheduler",
 "type" : "execute",
 "status" : "ok",
 "details" : {
 "status" : 200,
 "entity" : "{\n \"id\" : \"TwitterSearch\",\n \"dataSourceId\" :
\"TwitterSearch\",\n \"state\" : \"RUNNING\",\n \"message\" : null,\n \"startTime\"
: 1400254712000,\n \"endTime\" : -1,\n \"finished\" : false,\n \"counters\" : {
},\n \"exception\" : null,\n \"running\" : true\n}"
 } ]
}
```

Search Cluster API

The Search Cluster API lets you connect Fusion with existing Solr instances in a Zookeeper-managed cluster.

	Cluster operations are only supported when connecting through Zookeeper.

Once the Solr cluster is registered with Fusion, requests can be proxied through Fusion to it. The possible requests include search requests, but they can also be content indexing requests, such as the content crawled with a connector.

Once the searchCluster has been configured, the user can create Fusion collections that refer to the Solr collections that have been previously defined.

Background

Solr has three different approaches on how you can control visibility of new documents in search:

- You can `commit` manually
- You can rely on Solr's `autoCommit` setting
- You can specify `commitWithin` when adding documents

Fusion uses `commitWithin` to avoid relying on specific Solr side configurations. Fusion controls `commitWithin` on a per-collection basis so you can have multiple collections with different commit frequencies (for example, product documents can be committed more often than signals).

Global setting for `commitWithin`:

```
curl
https://localhost:8765/api/v1/configurations/com.lucidworks.apollo.solr.commitWithi
n 10000
```

`com.lucidworks.apollo.solr.commitWithin` is a global configuration property that defines default `commitWithin` for all documents added through Fusion. Every time you create a new collection in Fusion, per-collection `commitWithin` is initialized as the global default.

Per-collection setting: You can either specify this property when creating collection or update it with `PUT` later.

```
# create collection without specifying commitWithin sh> curl -H 'Content-type:
application/json' -X POST 'https://localhost:8765/api/v1/collections' -d '{"id" :
"test"}' { "id" : "test", ... "commitWithin" : 10000, ... } # create collection and
specify non default value sh> curl -H 'Content-type: application/json' -X POST
'https://localhost:8765/api/v1/collections' -d '{"id" : "test2", "commitWithin":
20000}' { "id" : "test2", ... "commitWithin" : 20000, ... } # update commitWithin
at a runtime sh> curl -H 'Content-type: application/json' -X PUT
'https://localhost:8765/api/v1/collections/test' -d ' { "id" : "test", "createdAt"
: "2015-01-07T17:44:47.396Z", "searchClusterId" : "default", "commitWithin" :
20000, "solrParams" : { "name" : "test", "numShards" : 1, "replicationFactor" : 1
}, "type" : "DATA", "metadata" : { } }'
```

Search Cluster Definition Properties

Property	Description
id *Required*	The ID of the search cluster. This is only required when creating a new cluster definition with a POST request.
connectString *Required*	The string to use to connect to the existing Solr cluster or standalone instance. If the existing Solr is running in SolrCloud mode, use the connect string for the ZooKeeper ensemble. If the existing Solr is running as a standalone instance, use the full URL for the Solr instance.
cloud *Required*	Defines if the "cluster" being defined is a SolrCloud cluster (**true**) or a standalone Solr instance (**false**).
bufferFlushInterval *Optional*	Defines how often to flush the buffer when writing to this cluster. If not defined, the system will default to 1000 milliseconds.
bufferSize *Optional*	Defines the size of the buffer. If not defined, the system will default to 100 items in the buffer.
concurrency *Optional*	Defines the maximum number of concurrent /parallel requests to Solr servers when Fusion index pipeline *Solr Indexer stage* has property bufferDocsForSolr set to true.
zkClientTimeout *Optional*	The maximum amount of time to wait when communicating with the ZooKeeper ensemble for a SolrCloud instance.
zkConnectTimeout *Optional*	The maximum amount of time to wait when trying to connect to the ZooKeeper ensemble for a SolrCloud instance.

Examples

Create a new search cluster that is an existing SolrCloud cluster:

REQUEST

```
curl -u USERNAME:PASSWORD -X POST -H 'Content-type: application/json' -d
'{"id":"mySolrCluster",
"connectString":"10.0.1.6:5001,10.0.1.6:5002,10.0.1.6:5003", "cloud":true}'
https://FUSION_HOST:FUSION_PORT/api/searchCluster
```

RESPONSE

```
{
  "id" : "mySolrCluster",
  "connectString" : "10.0.1.6:5001,10.0.1.6:5002,10.0.1.6:5003",
  "cloud" : true,
}
```

Create a 'cluster' that is a standalone Solr instance:

REQUEST

```
curl -u USERNAME:PASSWORD -X POST -H 'Content-type: application/json' -d
'{"id":"myOtherSolrCluster", "connectString":"https://FUSION_HOST:8983/solr",
"cloud":false}' https://FUSION_HOST:FUSION_PORT/api/searchCluster
```

RESPONSE

```
{
  "id" : "myOtherSolrCluster",
  "connectString" : "https://FUSION_HOST:8983/solr",
  "cloud" : false,
}
```

Show the status of each node of 'mySolrCluster':

REQUEST

```
curl https://FUSION_HOST:FUSION_PORT/api/searchCluster/mySolrCluster/nodes
```

RESPONSE

```
[ {
  "name" : "10.0.1.11:7574_solr",
  "baseUrl" : "https://10.0.1.11:7574/solr",
  "state" : "active"
}, {
  "name" : "10.0.1.8:7574_solr",
  "baseUrl" : "https://10.0.1.8:7574/solr",
  "state" : "active"
} ]
```

Show the system information for one named node:

REQUEST

```
curl https://10.0.1.8:FUSION_PORT/api/searchCluster/mySolrCluster/systemInfo?
nodeName=10.0.1.8:7574_solr
```

RESPONSE

```
{
  "10.0.1.8:7574_solr" : {
    "mode" : "solrcloud",
    "lucene" : {
      "solr-spec-version" : "4.8.0",
      "lucene-spec-version" : "4.8.0"
    },
    "jvm" : {
      "version" : "1.8.0_121 25.121-b13",
      "name" : "Oracle Corporation Java HotSpot(TM) 64-Bit Server VM",
      "processors" : 4,
      "memory" : {
        "raw" : {
          "free" : 66736272,
          "total" : 204800000,
          "max" : 204800000,
          "used" : 138063728,
          "used%" : 67.4139296875
        }
      }
    },
    "system" : {
      "name" : "Mac OS X",
      "version" : "10.9.3",
      "arch" : "x86_64",
      "systemLoadAverage" : 2.130859375,
      "committedVirtualMemorySize" : 2963378176,
      "freePhysicalMemorySize" : 9321914368,
      "freeSwapSpaceSize" : 1073741824,
      "processCpuTime" : 313176000000,
      "totalPhysicalMemorySize" : 17179869184,
      "totalSwapSpaceSize" : 1073741824,
      "openFileDescriptorCount" : 208,
      "maxFileDescriptorCount" : 10240,
      "uname" : "Darwin MacMini.local 13.2.0 Darwin Kernel Version 13.2.0: Thu Apr
17 23:03:13 PDT 2014; root:xnu-2422.100.13~1/RELEASE_X86_64 x86_64\n",
      "uptime" : "15:48  up 3 days,  7:08, 7 users, load averages: 2.13 2.01
1.91\n"
    }
  }
}
```

Solr API

The Solr API is used to manage collection-level configurations. The path for this request is:

```
https://FUSION_HOST:FUSION_PORT/api/solr/COLLECTION_NAME/SOLR_REQUEST
```

Replace `COLLECTION_NAME` with the name of a specific collection and `SOLR_REQUEST` with the Solr command you wish to run.

Since this API proxies requests to Solr, each available method corresponds to the method in Solr. So, a GET request to Solr would use the GET method of this endpoint; a POST would use the POST method, etc.

Depending on the request, the response may consist of records that match a query or output from a Schema API request.

Examples

Query Solr for documents in the 'docs' collection containing the term 'solr', limiting the results to only 2 records, returning only the title, and in JSON format:

REQUEST

```
https://FUSION_HOST:FUSION_PORT/api/solr/docs/select?
q=solr&rows=2&fl=title&wt=json
```

```
RESPONSE ---- { "responseHeader": { "status": 0, "QTime": 2, "params": { "fl":
"title", "q": "solr", "wt": "json", "rows": "2" } }, "response": { "numFound": 52,
"start": 0, "docs": [ { "title": [ "Solr and SolrAdmin APIs - Fusion Documentation
- Lucidworks" ] }, { "title": [ "Search Clusters - Fusion Documentation -
Lucidworks" ] } ] } } ----
```

Solr Configuration API

The Solr Configuration REST API lets you access and manage Solr configuration files stored in ZooKeeper.

To manage Solr lists of stopwords, see *Stopwords Files*.

Solr Configuration Definition Properties

Path Parameters

Parameter	Description
collection	The collection that contains the configuration files to list or view.
path	The path to a specific file or nested child nodes. If the file is not nested, the filename can be entered without any path information.

Query Parameters

Parameter	Description
expand	If true, the binary content of a file will be returned base64 encoded. If this is not included, only the metadata about each node will be returned. If you would like to see the content of the file in plain text, you can add 'Accept: text/plain' to the request header. Alternately, you can get the raw bytes by adding 'Accept: application/octet-stream' to the header.
recursive	If true, children of nested ZooKeeper nodes will be returned. This can be used in conjunction with the path to show only children of a specific node.

The output will include a list of each file with the following information:

- **name.** The name of the file or node.
- **version.** The file or node version from ZooKeeper.
- **isDir.** If the node has children, this value will be true.
- **value.** Only returned if "expand=true" is added as a query parameter. This will be returned as the base64 encoding of the contents of the file.

If the header includes 'Accept: text/plain' or 'Accept: application/octet-stream', metadata about the file will not be returned and only the content as either plain text or raw bytes.

If recursive=true and expand=true are both used in the same request, the request may be a little slow depending on how much data is requested.

ZooKeeper Collection Configuration Definition Properties

Path Parameters

Parameter	Description
collection	The collection that contains the configuration files to list or view.
path	The path to a specific file (including nested child nodes).

Query Parameters

Parameter	Description
reload	If true, the collection will be reloaded to make the changes available to Solr immediately.

Input Content

The REST API does not restrict the type of content that can be sent to ZooKeeper, but care should be taken to make sure the format of the file remains compatible with Solr's requirements.

As a best practice, all requests should include the Content-type in the request header.

Examples

Show _rest_managed.json for the 'docs' collection, with the base64 encoded content:

INPUT

```
curl -u USERNAME:PASSWORD
https://FUSION_HOST:FUSION_PORT/api/collections/docs/solr-
config/_rest_managed.json?expand=true
```

OUTPUT

```
{
  "name" : "_rest_managed.json",
  "version" : 0,
  "isDir" : false,
  "value" : "eyJpbml0QXJncyI6e30sIm1hbmFnZWRMaXN0IjpbXX0NCg=="
}
```

Get the plain text version of 'synonyms.txt':

INPUT

```
curl -u USERNAME:PASSWORD -H 'Accept: text/plain'
https://FUSION_HOST:FUSION_PORT/api/collections/docs/solr-config/synonyms.txt
```

OUTPUT

```
GB,gib,gigabyte,gigabytes
MB,mib,megabyte,megabytes
Television, Televisions, TV, TVs
pixima => pixma
```

Show the child nodes of 'clustering':

INPUT

```
curl -u USERNAME:PASSWORD
https://FUSION_HOST:FUSION_PORT/api/collections/docs/solr-config/clustering?
recursive=true
```

OUTPUT

```
{
  "name" : "clustering",
  "version" : 0,
  "isDir" : true,
  "children" : [ {
    "name" : "carrot2",
    "parent" : "clustering",
    "version" : 0,
    "isDir" : true,
    "children" : [ {
      "name" : "kmeans-attributes.xml",
      "parent" : "clustering/carrot2",
      "version" : 0,
      "isDir" : false
    }, {
      "name" : "lingo-attributes.xml",
      "parent" : "clustering/carrot2",
      "version" : 0,
      "isDir" : false
    }, {
      "name" : "stc-attributes.xml",
      "parent" : "clustering/carrot2",
      "version" : 0,
      "isDir" : false
    } ]
  } ]
}
```

Replace the contents of the existing synonyms.txt file with the contents of a file named "updated-synonyms.txt" and reload the collection:

INPUT

```
curl -u USERNAME:PASSWORD -X PUT -H 'Content-type: text/plain' --data-
binary'@updated-synonyms.txt'
https://FUSION_HOST:FUSION_PORT/api/collections/docs/solr-config/synonyms.txt?
reload=true
```

OUTPUT

None.

SolrAdmin API

The Solr Admin API lets you send commands to Solr through Fusion's proxy service. This allows you to protect your Solr instances from outside connections, and apply roles and user permissions from Fusion when running Solr commands. Requests sent to this API are subject to access restrictions above the collection level.

Note that because one searchCluster may host several collections, it is not recommended to use this with a collection-level command (such as a query, or document update).

Example

Here is an example to GET all CORE *MBeans* for the 'default' searchCluster, formatted in JSON:

REQUEST

```
https://FUSION_HOST:FUSION_PORT/api/solrAdmin/default/admin/mbeans?
cat=CORE&wt=json
```

RESPONSE

```
{ "responseHeader": { "status": 0, "QTime": 1 }, "solr-mbeans": [ "CORE", {
"searcher": { "class": "org.apache.solr.search.SolrIndexSearcher", "version":
"1.0", "description": "index searcher", "src": null }, "core": { "class":
"collection1", "version": "1.0", "description": "SolrCore", "src": null },
"Searcher@435cf502[collection1] main": { "class":
"org.apache.solr.search.SolrIndexSearcher", "version": "1.0", "description": "index
searcher", "src": null } } ] }
```

System API

The System REST API allows you to monitor the system performance.

Examples

Metric names

Get metric names that start with 'mem.heap':

REQUEST

```
curl -u USERNAME:PASSWORD https://FUSION_HOST:FUSION_PORT/api/system/metricNames?
prefix=mem.heap
```

RESPONSE

```
{
  "gauges" : [ "mem.heap.committed", "mem.heap.init", "mem.heap.max",
"mem.heap.usage", "mem.heap.used" ],
  "counters" : [ ],
  "histograms" : [ ],
  "meters" : [ ],
  "timers" : [ ]
}
```

Metrics

*Find all metrics that match the regular expression pattern 'com.lucidworks.apollo.**pipeline.index.**', format the response so it is readable, and show the sample data, if any:*

REQUEST

```
curl -u USERNAME:PASSWORD https://FUSION_HOST:FUSION_PORT/api/system/metrics?
pattern=com.lucidworks.apollo.*pipeline.index.*&pretty=true&showSamples=true
```

RESPONSE

```
{
  "version" : "3.0.0",
  "gauges" : { },
  "counters" : { },
  "histograms" : { },
  "meters" : {
    "com.lucidworks.apollo.pipeline.index.IndexPipelineCache.cacheHit" : {
      "count" : 4775,
      "m15_rate" : 0.03604340402401043,
      "m1_rate" : 0.04985610410800882,
      "m5_rate" : 0.04753263154077047,
      "mean_rate" : 0.05028487069705915,
      "units" : "events/second"
    }
  },
  "timers" : {
    "com.lucidworks.apollo.pipeline.index.IndexPipelineCache.deserialize" : {
      "count" : 7,
      "max" : 0.078783,
      "mean" : 0.028265285714285715,
      "min" : 9.800000000000001E-5,
      "p50" : 1.94E-4,
      "p75" : 0.06996100000000001,
      "p95" : 0.078783,
      "p98" : 0.078783,
      "p99" : 0.078783,
      "p999" : 0.078783,
      "values" : [ 9.800000000000001E-5, 1.09E-4, 1.62E-4, 1.94E-4, 0.04855,
0.06996100000000001, 0.078783 ],
      "stddev" : 0.03620774742010466,
      "m15_rate" : 2.964393875E-314,
      "m1_rate" : 2.128434034679706E-46,
      "m5_rate" : 4.9195401948202935E-138,
      "mean_rate" : 7.371603924510614E-5,
      "duration_units" : "seconds",
      "rate_units" : "calls/second"
    },
    "com.lucidworks.apollo.pipeline.index.IndexStageConfigCache.deserialize" : {
      "count" : 7,
      "max" : 0.002377,
      "mean" : 4.6642857142857147E-4,
      "min" : 8.7E-5,
      "p50" : 1.16E-4,
      "p75" : 3.46E-4,
      "p95" : 0.002377,
      "p98" : 0.002377,
      "p99" : 0.002377,
      "p999" : 0.002377,
      "values" : [ 8.7E-5, 9.300000000000001E-5, 1.11E-4, 1.16E-4, 1.35E-4, 3.46E-
4, 0.002377 ],
      "stddev" : 8.47267737523163E-4,
      "m15_rate" : 2.964393875E-314,
      "m1_rate" : 2.1522151745137967E-46,
      "m5_rate" : 5.086288568158318E-138,
      "mean_rate" : 7.372680363461613E-5,
      "duration_units" : "seconds",
      "rate_units" : "calls/second"
    },
    "com.lucidworks.apollo.pipeline.index.IndexStageConfigStore.deserialize" : {
      "count" : 6,
      "max" : 0.0019760000000000003,
      "mean" : 5.606666666666667E-4,
      "min" : 1.0800000000000001E-4,
      "p50" : 1.4250000000000002E-4,
      "p75" : 0.00114875,
```

```
      "p95" : 0.0019760000000000003,
      "p98" : 0.0019760000000000003,
      "p99" : 0.0019760000000000003,
      "p999" : 0.0019760000000000003,
      "values" : [ 1.0800000000000001E-4, 1.2200000000000001E-4, 1.35E-4,
1.5000000000000001E-4, 8.730000000000001E-4, 0.0019760000000000003 ],
      "stddev" : 7.54704622131511E-4,
      "m15_rate" : 2.964393875E-314,
      "m1_rate" : 1.0220227879692082E-48,
      "m5_rate" : 7.20591046931865E-140,
      "mean_rate" : 6.318494243486903E-5,
      "duration_units" : "seconds",
      "rate_units" : "calls/second"
    },
    "com.lucidworks.apollo.pipeline.index.IndexStageConfigStore.getItem" : {
      "count" : 6,
      "max" : 0.003128,
      "mean" : 0.002295,
      "min" : 0.0017770000000000002,
      "p50" : 0.0020715,
      "p75" : 0.002774,
      "p95" : 0.003128,
      "p98" : 0.003128,
      "p99" : 0.003128,
      "p999" : 0.003128,
      "values" : [ 0.0017770000000000002, 0.002066, 0.002066,
0.0020770000000000003, 0.0026560000000000004, 0.003128 ],
      "stddev" : 4.989869737778733E-4,
      "m15_rate" : 2.964393875E-314,
      "m1_rate" : 1.0220227879692082E-48,
      "m5_rate" : 7.20591046931865E-140,
      "mean_rate" : 6.318494244418448E-5,
      "duration_units" : "seconds",
      "rate_units" : "calls/second"
    }
  }
}
```


The above output has been truncated for space to remove metrics with no data or with very long value lists.

Threads

REQUEST

```
curl -u USERNAME:PASSWORD https://FUSION_HOST:FUSION_PORT/api/system/threads
```

RESPONSE (truncated to a single thread)

```
[ {
  "id" : 2,
  "native" : false,
  "name" : "Reference Handler",
  "locks" : {
    "waiting" : {
      "identity" : "0x7257d934",
      "class" : "java.lang.ref.Reference$Lock"
    },
    "locking" : {
      "identity" : "0x7257d934",
      "class" : "java.lang.ref.Reference$Lock"
    }
  },
  "state" : "WAITING",
  "suspended" : false,
  "stackTrace" : [ {
    "methodName" : "wait",
    "fileName" : "Object.java",
    "lineNumber" : -2,
    "className" : "java.lang.Object",
    "nativeMethod" : true
  }, {
    "methodName" : "wait",
    "fileName" : "Object.java",
    "lineNumber" : 503,
    "className" : "java.lang.Object",
    "nativeMethod" : false
  }, {
    "methodName" : "run",
    "fileName" : "Reference.java",
    "lineNumber" : 133,
    "className" : "java.lang.ref.Reference$ReferenceHandler",
    "nativeMethod" : false
  } ]
} ]
```


Buffers

REQUEST

```
curl -u USERNAME:PASSWORD -X PUT
https://FUSION_HOST:FUSION_PORT/api/system/buffers
```

Ping

REQUEST

```
curl -u USERNAME:PASSWORD https://FUSION_HOST:FUSION_PORT/api/system/ping
```

RESPONSE

```
pong
```

ZooKeeper Import/Export API

Apache ZooKeeper *(https://zookeeper.apache.org/)* is a distributed configuration service, synchronization service, and naming registry. Fusion uses ZooKeeper to configure and manage all Fusion components in a single Fusion deployment.

The ZooKeeper Import/Export API provides methods to upload or download information from Fusion's ZooKeeper service. This service provides an alternative to the ZooKeeper clients `zkCli.sh` and `zk-shell` which are part of the Apache Zookeeper distribution included as part of the Fusion distribution.

The ZKImportExport service may be used to export ZooKeeper data for any Fusion release. It can be used to import configuration data into the ZooKeeper service for a new or existing Fusion deployment.

- For details on using this script during the Fusion upgrade procedure, see *Upgrading Fusion*.
- For details on using this script to migrate Fusion configurations from one deployment to another, see *Migrating Fusion Objects by Hand*.

The REST API only supports requests to export ZooKeeper configurations. The Fusion distribution includes a utility script `zkImportExport.sh` which can be used to import ZooKeeper configuration as well as to export it from arbitrary Fusion instances.

See the official ZooKeeper documentation *(https://zookeeper.apache.org/doc/current/)* for details about using and managing a ZooKeeper service.

Utility script `zkImportExport.sh`

This script is located in the top-level Fusion `scripts` directory. The script takes the following command-line arguments:

```
-c,--cmd <arg>        Command, one of: 'export', 'import', 'update', 'delete'.

-e,--encode <arg>     Type of encoding for znodes. Valid options:
                       'none', 'utf-8', 'base64', default is 'base64'.
                       Option 'none' will not return any data from the znodes.

-ep,--exclude <arg>   Exclude znode paths, followed by list of paths.
                      Can only be used to exclude nodes one level below the root
node.

-eph,--ephemeral      Include ephemeral nodes while exporting znodes, boolean,
default false.

-f,--filename <arg>   Name of file containing import/export data.

-h,--help             Display help page.

-ip,--include <arg>   Include znode paths to include, followed by a list of paths.
                      Can only be used to include nodes one level below the root
node.

-nr,--non-recursive   Do not perform recursive operations on znodes.

-o,--overwrite        Overwrite data for existing znodes. Valid only with 'update'
command.

-p,--path <arg>       Path from ZooKeeper root node, e.g. '/lucid/query-pipelines'.

-r,--recursive        Perform recursive operations on znodes.

-z,--zkhost <arg>     ZooKeeper Connect string, required.
```

Required arguments are: * -c, --cmd : operation to perform. * -z, --zkhost : the ZooKeeper Connect string.

Examples

Export all data from a local single-node ZooKeeper service, save data to a file:

```
zkImportExport.sh -zkhost FUSION_HOST:9983 -cmd export -path / -filename
znode_dump.json
```

Export all Fusion configurations from a local single-node ZooKeeper service, save data to a file:

```
zkImportExport.sh -zkhost FUSION_HOST:9983 -cmd export -path /lucid -filename
znode_lucid_dump.json
```

Export Fusion user databases, groups, roles, and realms configurations from a local single-node ZooKeeper service, save data to a file:

```
zkImportExport.sh -zkhost FUSION_HOST:9983 -cmd export -path /lucid-apollo-admin -
filename znode_lucid_admin_dump.json
```

Initial import of saved Fusion configuration into a new ZooKeeper:

```
zkImportExport.sh -zkhost FUSION_HOST:9983 -cmd import -filename
znode_lucid_dump.json
```

Note that the above command will fail if there is conflict between existing znode structures or contents between the ZooKeeper service and the dump file.

Update information for Fusion's ZooKeeper service:

```
zkImportExport.sh -zkhost FUSION_HOST:9983 -cmd update -filename
znode_lucid_dump.json
```

Remove a znode from Fusion's ZooKeeper service:

```
zkImportExport.sh -zkhost FUSION_HOST:9983 -cmd delete -path /lucid/test
```

Fusion REST API service ZKImportExport

The Fusion REST API can only be used to download information from ZooKeeper, via the 'GET' method with the following configuration:

```
  "zk-import-export::v1" : {
    "name" : "com.lucidworks.apollo.resources.ZKImportExportResource",
    "uri" : "/zk/export",
    "methods" : [ {
      "uri" : "/zk/export/{path:.*}",
      "name" : "getNodeInfo",
      "verb" : "GET",
      "pathParams" : [ {
        "name" : "path",
        "type" : "String"
      } ],
      "queryParams" : [ {
        "name" : "recursive",
        "type" : "Boolean"
      }, {
        "name" : "excludePaths",
        "type" : "List"
      }, {
        "name" : "includePaths",
        "type" : "List"
      }, {
        "name" : "encodeValues",
        "type" : "String"
      } ],
      "hasEntity" : false
    } ]
  }
```

GET data from path `/lucid/query-pipelines`

```
curl -u USERNAME:PASSWORD -X GET
https://FUSION_HOST:FUSION_PORT/api/zk/export/lucid/query-pipelines { "path" :
"/lucid/query-pipelines", "children" : [ { "path" : "/lucid/query-
pipelines/default", "data" :
"ewogICJpZCIgOiAiZGVmYXVsdCIsCiAgInN0YWdlcyIgOiBbIHsKICAgICJ0eXBlIiA6ICJzZWFyY2gtZm
llbGRzIiwKICAgICJpZCIgOiAiMzc1NmI1ZDctY2MwMC00MDAyLWJiOWQtNTQzNjQ4NzVjMjgyIiwKICAgI
CJyb3dzIiA6IDEwLAogICAgInN0YXJ0IiA6IDAsCiAgICAic2tpcCIgOiBmYWxzZSwKICAgICJsYWJlbCIg
OiAic2VhcmNoLWZpZWxkcyIsCiAgICAidHlwZSIgOiAic2VhcmNoLWZpZWxkcyIKICB9LCB7CiAgICAidHl
wZSIgOiAiZmFjZXQiLAogICAgImlkIiA6ICI3MTFmZWM1Ni03MzRmLTRiYTAtOWY1NS0wZTQ4YmU2NTllM2
UiLAogICAgInNraXAiIDogZmFsc2UsCiAgICAibGFiZWwiIDogImZhY2V0IiwKICAgICJ0eXBlIiA6ICJmY
WNldCIKICB9LCB7CiAgICAidHlwZSIgOiAic29sci1xdWVyeSIsCiAgICAiaWQiIDogIjYwNDU1YjU2LTdk
N2MtNDZjYi1iN2QzLTIxOWU1N2U3MWNjMyIsCiAgICAiaHR0cE1ldGhvZCIgOiAiUE9TVCIsCiAgICAic2t
pcCIgOiBmYWxzZSwKICAgICJsYWJlbCIgOiAic29sci1xdWVyeSIsCiAgICAidHlwZSIgOiAic29sci1xdW
VyeSIKICB9IF0KfQ==" } ], "data" : "" }
```

Get info for node `/lucid/query-pipelines`. Do not expand the znodes

```
curl -u USERNAME:PASSWORD -X GET
https://FUSION_HOST:FUSION_PORT/api/zk/export/lucid/query-pipelines?
recursive=true&encodeValues=none { "path" : "/lucid/query-pipelines", "children" :
[ { "path" : "/lucid/query-pipelines/default" } ] }
```

Get info for node `/lucid/query-pipelines`. encode in `utf-8`

```
curl -u USERNAME:PASSWORD -X GET
https://FUSION_HOST:FUSION_PORT/api/zk/export/lucid/query-pipelines?
recursive=true&encodeValues=utf-8 { "path" : "/lucid/query-pipelines", "children" :
[ { "path" : "/lucid/query-pipelines/default", "data" : "{\n \"id\" :
\"default\",\n \"stages\" : [ {\n \"type\" : \"search-fields\",\n \"id\" :
\"3756b5d7-cc00-4002-bb9d-54364875c282\",\n \"rows\" : 10,\n \"start\" : 0,\n
\"skip\" : false,\n \"label\" : \"search-fields\",\n \"type\" : \"search-fields\"\n
}, {\n \"type\" : \"facet\",\n \"id\" : \"711fec56-734f-4ba0-9f55-0e48be659e3e\",\n
\"skip\" : false,\n \"label\" : \"facet\",\n \"type\" : \"facet\"\n }, {\n \"type\"
: \"solr-query\",\n \"id\" : \"60455b56-7d7c-46cb-b7d3-219e57e71cc3\",\n
\"httpMethod\" : \"POST\",\n \"skip\" : false,\n \"label\" : \"solr-query\",\n
\"type\" : \"solr-query\"\n } ]\n}" } ], "data" : "" }
```

Get info for node `/lucid/query-pipelines`. *Exclude path `/lucid/query-pipelines/default`*

```
curl -u USERNAME:PASSWORD -X GET
https://FUSION_HOST:FUSION_PORT/api/zk/export/lucid/query-pipelines?
recursive=true&encodeValues=utf-8&excludePaths=/lucid/query-piplines/default {
"path" : "/lucid/query-pipelines", "data" : "" }
```

Query Service APIs

The Query Service APIs are a group of APIs that allow you to customize your search experience. You can use them to *run queries*, *reorder and test pipeline stages*, *improve search relevancy*, and more.

Query Rewrite API Configurations

The following topics contain parameters and other necessary information to configure the corresponding rule or rewrite type for the *Query Rewrite API*.

Rule configurations

Banner Rule API Configurations

Block List Rule API Configurations

Boost Attributes Rule API Configurations

Boost List Rule API Configurations

Bury List Rule API Configurations

Filter List Rule API Configurations

Ingroup Boost List Rule API Configurations

Ingroup Bury List Rule API Configurations

Ingroup Pinned List Rule API Configurations

JSON Blob Rule API Configurations

Pinned Rule API Configurations

Redirect Rule API Configurations

Response Value Rule API Configurations

Set Facets Rule API Configurations

Set Params Rule API Configurations

Query rewrite configurations

Ground Truth Query Rewrite API Configurations

Phrase Query Rewrite API Configurations

Remove Words Query Rewrite API Configurations

Similar Queries (simq) Query Rewrite API Configurations

Spell Query Rewrite API Configurations

Synonym Query Rewrite API Configurations

Tail Query Rewrite API Configurations

Query Rewrite API

The Query Rewrite API manages documents used for *query rewriting* to improve relevancy using AI-generated data.

Query Rewrite Rules Configuration

To access detailed information about configuring the rewrite functions and rules for the Query Rewrite API, see *Query Rewrite API Configurations*.

Examples

View configuration options

To see full configuration options using the Query Rewrite API:

- **All rules.** `https://FUSION_HOST:FUSION_PORT/api/apps/APP_NAME/query-rewrite/schema`
- **Individual rules.** `https://FUSION_HOST:FUSION_PORT/api/apps/APP_NAME/query-rewrite/schema/RULE_TYPE`

> The `RULE_TYPE` values are found at the **.../query-rewrite/schema** endpoint: `oneOf` > `0` > `properties` > `type` > `default`.

```
{ "pinned_values": { "type": "array", "title": "Pinned Documents", "description":
"A list of one or more documents to boost and pin to specific positions in the
results (1-based index). Note that pinning will only be applied to the first page
of results.", "minItems": 1, "items": { "type": "object", "required": ["value",
"position"], "properties": { "value": { "type": "string", "title": "Value",
"description": "Value to boost" }, "position": { "type": "integer", "title":
"Position", "description": "Absolute position in the first page of results (1-based
index)" } } }}},
```

Create a blocklist rule

```
curl -u USERNAME:PASSWORD -XPOST -H "Content-type:application/json"
https://FUSION_HOST:FUSION_PORT/api/apps/APP_NAME/query-rewrite/instances -d '{
"id": "block_list_rule_1", "name": "block_list_rule_1", "type": "block_list",
"field_name": "id", "field_values": ["6","17"], "enabled": true, "priority":11,
"search_terms":["life"], "matching": "keywords" }'
```

Update specific fields in an existing rule

```
curl -u USERNAME:PASSWORD -XPUT -H "Content-type:application/json"
https://FUSION_HOST:FUSION_PORT/api/apps/APP_NAME/query-rewrite/instances -d '{
"id": "block_list_rule_1", "name": "block_list_rule_1", "type": "block_list",
"field_name": "id", "field_values": ["6","17"], "enabled": false, "priority":9,
"search_terms":["life"], "matching": "keywords" }'
```

> When making a PUT request to `query-rewrite/instances` include as a body JSON `{"create":[],"update":[],"delete":[],}` to avoid errors for missing properties.

Publish rules in the staging area

```
curl -u USERNAME:PASSWORD -X PUT -H 'Content-Type: application/json'
https://FUSION_HOST:FUSION_PORT/api/apps/APP_NAME/query-rewrite/publish
```

Delete a published rule

```
curl -u USERNAME:PASSWORD -XPUT -H "Content-type:application/json"
https://FUSION_HOST:FUSION_PORT/api/apps/APP_NAME/query-rewrite/instances -d '{
"delete": [ “block_list_rule_1" ] }'
```

Publish an operation

```
curl -u USERNAME:PASSWORD -XPUT -H "Content-type:application/json"
https://FUSION_HOST:FUSION_PORT/api/apps/APP_NAME/query-rewrite/publish
```

Banner Rule API Configurations

This rule type is used in the *Query Rewrite API*.

For a complete list of rule and rewrite types in the Query Rewrite API, see *Query Rewrite API Configurations*.

Banner

The banner rule can add a banner URL to the response before returning it to the client.

General information

- Endpoint: `https://FUSION_HOST:FUSION_PORT/api/apps/APP_NAME/query-rewrite/schema/RULE_TYPE`
- Required parameters are:
  - name
  - banner_url
  - type
- The values for the `hints` parameter include:
  - `hidden`: Do not display in the Fusion UI, but can be read and written in the schema.
  - `readonly`: Cannot be set by the user. If attempted, an error is generated and the operation fails.

Parameter	Schema
type	
type	banner
properties	
id	`"id": { "type": "string", "title": "ID", "description": "The ID for this Rule. Used in the API to reference this query rewrite rule. Allowed characters: a-z, A-Z, 0-9, dash (-) and underscore (_)", "maxLength": 128, "pattern": "^[A-Za-z0-9_\\-]+$" }`
source	`"source": { "type": "string", "title": "Source", "description": "The source for this query rewrite rule, such as a Spark Job.", "hints": [ "hidden" ] }`
sourceType	`"sourceType": { "type": "string", "title": "Source Type", "description": "Source type", "hints": [ "hidden" ] }`
createdOn	`"createdOn": { "type": "string", "title": "Created Date/Time", "description": "Date/Time this rule was created.", "format": "date-time" }`
updatedOn	`"updatedOn": { "type": "string", "title": "Updated Date/Time", "description": "Date/Time this rule was last updated.", "format": "date-time" }`

Parameter	Schema
reviewer	`"reviewer": { "type": "string", "title": "Reviewer", "description": "Who reviewed this rewrite rule.", "hints": [ "hidden" ] }`
reviewedOn	`"reviewedOn": { "type": "string", "title": "Reviewed Date/Time", "description": "Date/Time this rule was last reviewed.", "hints": [ "hidden" ], "format": "date-time" }`
review	`"review": { "type": "string", "title": "Review", "enum": [ "approved", "denied" ], "default": "approved", "hints": [ "readonly" ] }`
deployed	`"deployed": { "type": "boolean", "title": "Deployed Flag", "description": "Flag to indicate if this rewrite rule has been deployed.", "hints": [ "hidden" ] }`
tags	`"tags": { "type": "array", "title": "Tags", "description": "Conditional tags to restrict the triggering of this query rewrite rule.", "items": { "type": "string" } }`
name Required	`"name": { "type": "string", "title": "Name", "description": "User-friendly name for this rule." }`

Parameter	Schema
description	`"description": { "type": "string", "title": "Description", "description": "Short description of what this rule does." }`
enabled	`"enabled": { "type": "boolean", "title": "Enabled", "description": "Set to false to disable this rule.", "default": true }`
matching	`"matching": { "type": "string", "title": "Match Query Using", "description": "Defines how the specified search terms will be compared to the user's query to determine whether this rule applies. 'keywords' means the query must match the keywords of the search terms value exactly. 'phrase' means the search query must contain one of the search terms values as a subphrase. 'text' means that at least one of the search terms values must match the query, but unlike 'phrase', the query only needs to match part of the value.", "enum": ["keywords", "text", "phrase"], "default": "keywords" }`
groups	`"groups": { "type": "string", "title": "Rule Group", "description": "Group for this rule." }`
effective_range	`"effective_range": { "type": "array", "title": "Effective Range", "description": "Time range when this rule is in effect.", "items": { "type": "string" } }`

Parameter	Schema
priority	"priority": { "type": "integer", "title": "Precedence", "description": "Used when multiple rules could take effect for a given query - higher number means higher precedence. For example, a rule with precedence of 10 will take effect before a rule with precedence of 3. Ties on precedence are broken based on creation time - the more recently created rule will take effect before the older rule." }
search_terms	"search_terms": { "type": "array", "title": "Search Terms", "description": "Search terms that trigger this rule.", "items": { "type": "string" } }
filters	"filters": { "type": "array", "title": "Filters", "description": "Filters used to determine if this rule triggers", "items": { "type": "object", "required": ["key", "value"], "properties": { "key": { "type": "string", "title": "Field Name" }, "value": { "type": "string", "title": "Field Value" } } } }
response_values	"response_values": { "type": "array", "title": "Set Response Values", "description": "Key/Value pairs to set in the fusion section of the response.", "hints": ["hidden"], "items": { "type": "object", "required": ["key", "value"], "properties": { "key": { "type": "string", "title": "Field Name" }, "value": { "type": "string", "title": "Field Value" } } } }
query_profile_id_ss	"query_profile_id_ss": { "type": "array", "title": "Query Profile ID", "description": "Query Profile that triggers this rule", "items": { "type": "string" } }

Parameter	Schema
banner_url Required	`"banner_url": { "type": "string", "title": "Banner URL", "description": "URL to a banner, typically an HTML file" }`
zone	`"zone": { "type": "string", "title": "Banner Zone", "description": "Zone for the banner in the search UI" }`
type Required	`"type": { "type": "string", "title": "Type", "enum": [ "banner" ], "default": "banner", "hints": [ "readonly" ] }`
doc_type	`"doc_type": { "type": "string", "title": "Doc Type", "enum": [ "rule" ], "default": "rule", "hints": [ "readonly" ] }`
display_type	`"display_type": { "type": "string", "title": "Display Type", "enum": [ "Banner" ], "default": "Banner", "hints": [ "readonly" ] } }`
additionalProperties	
additionalProperties	`"additionalProperties": true`

Block List Rule API Configurations

This rule type is used in the *Query Rewrite API*.

For a complete list of rule and rewrite types in the Query Rewrite API, see *Query Rewrite API Configurations*.

Block List

The Block List action blocks the associated values from appearing in the search results.

For example, a workplace may block documents that the user does not have permission to view.

General information

- Endpoint: `https://FUSION_HOST:FUSION_PORT/api/apps/APP_NAME/query-rewrite/schema/RULE_TYPE`
- Required parameters are:
  - name
  - field_name
  - field_values
  - type
- The values for the `hints` parameter include:
  - `hidden`: Do not display in the Fusion UI, but can be read and written in the schema.
  - `readonly`: Cannot be set by the user. If attempted, an error is generated and the operation fails.

Parameter	Schema
type	
type	string
properties	
id	`"id": { "type": "string", "title": "ID", "description": "The ID for this Rule. Used in the API to reference this query rewrite rule. Allowed characters: a-z, A-Z, 0-9, dash (-) and underscore (_)", "maxLength": 128, "pattern": "^[A-Za-z0-9_\\-]+$" }`
source	`"source": { "type": "string", "title": "Source", "description": "The source for this query rewrite rule, such as a Spark Job.", "hints": [ "hidden" ] }`
sourceType	`"sourceType": { "type": "string", "title": "Source Type", "description": "Source type", "hints": [ "hidden" ] }`
createdOn	`"createdOn": { "type": "string", "title": "Created Date/Time", "description": "Date/Time this rule was created.", "format": "date-time" }`
updatedOn	`"updatedOn": { "type": "string", "title": "Updated Date/Time", "description": "Date/Time this rule was last updated.", "format": "date-time" }`

Parameter	Schema
reviewer	`"reviewer": { "type": "string", "title": "Reviewer", "description": "Who reviewed this rewrite rule.", "hints": [ "hidden" ] }`
reviewedOn	`"reviewedOn": { "type": "string", "title": "Reviewed Date/Time", "description": "Date/Time this rule was last reviewed.", "hints": [ "hidden" ], "format": "date-time" }`
review	`"review": { "type": "string", "title": "Review", "enum": [ "approved", "denied" ], "default": "approved", "hints": [ "readonly" ] }`
deployed	`"deployed": { "type": "boolean", "title": "Deployed Flag", "description": "Flag to indicate if this rewrite rule has been deployed.", "hints": [ "hidden" ] }`
tags	`"tags": { "type": "array", "title": "Tags", "description": "Conditional tags to restrict the triggering of this query rewrite rule.", "items": { "type": "string" } }`
name Required	`"name": { "type": "string", "title": "Name", "description": "User-friendly name for this rule." }`

Parameter	Schema
description	`"description": { "type": "string", "title": "Description", "description": "Short description of what this rule does." }`
enabled	`"enabled": { "type": "boolean", "title": "Enabled", "description": "Set to false to disable this rule.", "default": true }`
matching	`"matching": { "type": "string", "title": "Match Query Using", "description": "Defines how the specified search terms will be compared to the user's query to determine whether this rule applies. 'keywords' means the query must match the keywords of the search terms value exactly. 'phrase' means the search query must contain one of the search terms values as a subphrase. 'text' means that at least one of the search terms values must match the query, but unlike 'phrase', the query only needs to match part of the value.", "enum": ["keywords", "text", "phrase"], "default": "keywords" }`
groups	`"groups": { "type": "string", "title": "Rule Group", "description": "Group for this rule." }`
effective_range	`"effective_range": { "type": "array", "title": "Effective Range", "description": "Time range when this rule is in effect.", "items": { "type": "string" } }`

Parameter	Schema
priority	"priority": { "type": "integer", "title": "Precedence", "description": "Used when multiple rules could take effect for a given query - higher number means higher precedence. For example, a rule with precedence of 10 will take effect before a rule with precedence of 3. Ties on precedence are broken based on creation time - the more recently created rule will take effect before the older rule." }
search_terms	"search_terms": { "type": "array", "title": "Search Terms", "description": "Search terms that trigger this rule.", "items": { "type": "string" } }
filters	"filters": { "type": "array", "title": "Filters", "description": "Filters used to determine if this rule triggers", "items": { "type": "object", "required": ["key", "value"], "properties": { "key": { "type": "string", "title": "Field Name" }, "value": { "type": "string", "title": "Field Value" } } } }
response_values	"response_values": { "type": "array", "title": "Set Response Values", "description": "Key/Value pairs to set in the fusion section of the response.", "items": { "type": "object", "required": ["key", "value"], "properties": { "key": { "type": "string", "title": "Field Name" }, "value": { "type": "string", "title": "Field Value" } } } }
query_profile_id_ss	"query_profile_id_ss": { "type": "array", "title": "Query Profile ID", "description": "Query Profile that triggers this rule", "items": { "type": "string" } }

Parameter	Schema
field_name Required	"field_name": { "type": "string", "title": "Field Name", "description": "Set the field name used by this rule to match the specified values.", "default": "id" }
field_values Required	"field_values": { "type": "array", "title": "Field Values", "description": "List of values for the specified field.", "minItems": 1, "items": { "type": "string" } }
terms_query_separator_s	"terms_query_separator_s": { "type": "string", "title": "Terms Query Separator", "description": "Separator delimiting the values in the terms query string, defaulting to a comma.", "default": ",", "hints": ["hidden"] }
terms_query_method_s	"terms_query_method_s": { "type": "string", "title": "Terms Query Method", "description": "One of termsFilter (default), booleanQuery, automaton, docValuesTermsFilter", "default": "termsFilter", "hints": ["hidden"] }
type Required	"type": { "type": "string", "title": "Type", "enum": ["block_list"], "default": "block_list", "hints": ["readonly"] }
doc_type	"doc_type": { "type": "string", "title": "Doc Type", "enum": ["rule"], "default": "rule", "hints": ["readonly"] }

Parameter	Schema
display_type	`"display_type": { "type": "string", "title": "Display Type", "enum": [ "Block List" ], "default": "Block List", "hints": [ "readonly" ] } }`
additionalProperties	
additionalProperties	`"additionalProperties": true`

Boost Attributes Rule API Configurations

This rule type is used in the *Query Rewrite API*.

For a complete list of rule and rewrite types in the Query Rewrite API, see *Query Rewrite API Configurations*.

Boost Attributes

The Boost Attributes action boosts documents with specific attributes in a search. Boosts can be additive or multiplicative.

For example, an e-commerce website may boost the laptop manufacturer's name if the search query contains a manufacturer.

General information

- Endpoint: `https://FUSION_HOST:FUSION_PORT/api/apps/APP_NAME/query-rewrite/schema/RULE_TYPE`
- Required parameters are:
  - name
  - boost_param
  - boost_query
  - type
- The values for the `hints` parameter include:
  - `hidden`: Do not display in the Fusion UI, but can be read and written in the schema.
  - `readonly`: Cannot be set by the user. If attempted, an error is generated and the operation fails.

Parameter	Schema
type	
type	string
properties	
id	`"id": { "type": "string", "title": "ID", "description": "The ID for this Rule. Used in the API to reference this query rewrite rule. Allowed characters: a-z, A-Z, 0-9, dash (-) and underscore (_)", "maxLength": 128, "pattern": "^[A-Za-z0-9_\\-]+$" }`
source	`"source": { "type": "string", "title": "Source", "description": "The source for this query rewrite rule, such as a Spark Job.", "hints": [ "hidden" ] }`
sourceType	`"sourceType": { "type": "string", "title": "Source Type", "description": "Source type", "hints": [ "hidden" ] }`
createdOn	`"createdOn": { "type": "string", "title": "Created Date/Time", "description": "Date/Time this rule was created.", "format": "date-time" }`
updatedOn	`"updatedOn": { "type": "string", "title": "Updated Date/Time", "description": "Date/Time this rule was last updated.", "format": "date-time" }`

Parameter	Schema
reviewer	`"reviewer": { "type": "string", "title": "Reviewer", "description": "Who reviewed this rewrite rule.", "hints": [ "hidden" ] }`
reviewedOn	`"reviewedOn": { "type": "string", "title": "Reviewed Date/Time", "description": "Date/Time this rule was last reviewed.", "hints": [ "hidden" ], "format": "date-time" }`
review	`"review": { "type": "string", "title": "Review", "enum": [ "approved", "denied" ], "default": "approved", "hints": [ "readonly" ] }`
deployed	`"deployed": { "type": "boolean", "title": "Deployed Flag", "description": "Flag to indicate if this rewrite rule has been deployed.", "hints": [ "hidden" ] }`
tags	`"tags": { "type": "array", "title": "Tags", "description": "Conditional tags to restrict the triggering of this query rewrite rule.", "items": { "type": "string" } }`
name Required	`"name": { "type": "string", "title": "Name", "description": "User-friendly name for this rule." }`

Parameter	Schema
description	`"description": { "type": "string", "title": "Description", "description": "Short description of what this rule does." }`
enabled	`"enabled": { "type": "boolean", "title": "Enabled", "description": "Set to false to disable this rule.", "default": true }`
matching	`"matching": { "type": "string", "title": "Match Query Using", "description": "Defines how the specified search terms will be compared to the user's query to determine whether this rule applies. 'keywords' means the query must match the keywords of the search terms value exactly. 'phrase' means the search query must contain one of the search terms values as a subphrase. 'text' means that at least one of the search terms values must match the query, but unlike 'phrase', the query only needs to match part of the value.", "enum": ["keywords", "text", "phrase"], "default": "keywords" }`
groups	`"groups": { "type": "string", "title": "Rule Group", "description": "Group for this rule." }`
effective_range	`"effective_range": { "type": "array", "title": "Effective Range", "description": "Time range when this rule is in effect.", "items": { "type": "string" } }`

Parameter	Schema
priority	"priority": { "type": "integer", "title": "Precedence", "description": "Used when multiple rules could take effect for a given query - higher number means higher precedence. For example, a rule with precedence of 10 will take effect before a rule with precedence of 3. Ties on precedence are broken based on creation time - the more recently created rule will take effect before the older rule." }
search_terms	"search_terms": { "type": "array", "title": "Search Terms", "description": "Search terms that trigger this rule.", "items": { "type": "string" } }
filters	"filters": { "type": "array", "title": "Filters", "description": "Filters used to determine if this rule triggers", "items": { "type": "object", "required": ["key", "value"], "properties": { "key": { "type": "string", "title": "Field Name" }, "value": { "type": "string", "title": "Field Value" } } } }
response_values	"response_values": { "type": "array", "title": "Set Response Values", "description": "Key/Value pairs to set in the fusion section of the response.", "items": { "type": "object", "required": ["key", "value"], "properties": { "key": { "type": "string", "title": "Field Name" }, "value": { "type": "string", "title": "Field Value" } } } }
query_profile_id_ss	"query_profile_id_ss": { "type": "array", "title": "Query Profile ID", "description": "Query Profile that triggers this rule", "items": { "type": "string" } }

Parameter	Schema
boost_param Required	"boost_param": { "type": "string", "title": "Boost Param", "description": "The boost param to use, either: bq (additive) or boost (multiplicative). Only additive boosts are supported for requests using the query DSL", "default": "bq" }
boost_query Required	"boost_query": { "type": "string", "title": "Boost Query", "description": "The boost query" }
type Required	"type": { "type": "string", "title": "Type", "enum": ["boost_attributes"], "default": "boost_attributes", "hints": ["readonly"] }
doc_type	"doc_type": { "type": "string", "title": "Doc Type", "enum": ["rule"], "default": "rule", "hints": ["readonly"] }
display_type	"display_type": { "type": "string", "title": "Display Type", "enum": ["Boost Attributes"], "default": "Boost Attributes", "hints": ["readonly"] } }
additionalProperties	
additionalProperties	"additionalProperties": true

Boost List Rule API Configurations

This rule type is used in the *Query Rewrite API*.

For a complete list of rule and rewrite types in the Query Rewrite API, see *Query Rewrite API Configurations*.

Boost List

The Boost List action boosts documents with a large multiplicative boost so the documents appear at the top of the results. Larger boosts can override this action.

For example, an e-commerce website may boost new or seasonal items.

General information

- Endpoint: `https://FUSION_HOST:FUSION_PORT/api/apps/APP_NAME/query-rewrite/schema/RULE_TYPE`
- Required parameters are:
  - name
  - field_name
  - field_values
  - type
- The values for the `hints` parameter include:
  - `hidden`: Do not display in the Fusion UI, but can be read and written in the schema.
  - `readonly`: Cannot be set by the user. If attempted, an error is generated and the operation fails.

Parameter	Schema
type	
type	string
properties	
id	`"id": { "type": "string", "title": "ID", "description": "The ID for this Rule. Used in the API to reference this query rewrite rule. Allowed characters: a-z, A-Z, 0-9, dash (-) and underscore (_)", "maxLength": 128, "pattern": "^[A-Za-z0-9_\\-]+$" }`
source	`"source": { "type": "string", "title": "Source", "description": "The source for this query rewrite rule, such as a Spark Job.", "hints": [ "hidden" ] }`
sourceType	`"sourceType": { "type": "string", "title": "Source Type", "description": "Source type", "hints": [ "hidden" ] }`
createdOn	`"createdOn": { "type": "string", "title": "Created Date/Time", "description": "Date/Time this rule was created.", "format": "date-time" }`
updatedOn	`"updatedOn": { "type": "string", "title": "Updated Date/Time", "description": "Date/Time this rule was last updated.", "format": "date-time" }`

Parameter	Schema
reviewer	`"reviewer": { "type": "string", "title": "Reviewer", "description": "Who reviewed this rewrite rule.", "hints": [ "hidden" ] }`
reviewedOn	`"reviewedOn": { "type": "string", "title": "Reviewed Date/Time", "description": "Date/Time this rule was last reviewed.", "hints": [ "hidden" ], "format": "date-time" }`
review	`"review": { "type": "string", "title": "Review", "enum": [ "approved", "denied" ], "default": "approved", "hints": [ "readonly" ] }`
deployed	`"deployed": { "type": "boolean", "title": "Deployed Flag", "description": "Flag to indicate if this rewrite rule has been deployed.", "hints": [ "hidden" ] }`
tags	`"tags": { "type": "array", "title": "Tags", "description": "Conditional tags to restrict the triggering of this query rewrite rule.", "items": { "type": "string" } }`
name Required	`"name": { "type": "string", "title": "Name", "description": "User-friendly name for this rule." }`

Parameter	Schema
description	`"description": { "type": "string", "title": "Description", "description": "Short description of what this rule does." }`
enabled	`"enabled": { "type": "boolean", "title": "Enabled", "description": "Set to false to disable this rule.", "default": true }`
matching	`"matching": { "type": "string", "title": "Match Query Using", "description": "Defines how the specified search terms will be compared to the user's query to determine whether this rule applies. 'keywords' means the query must match the keywords of the search terms value exactly. 'phrase' means the search query must contain one of the search terms values as a subphrase. 'text' means that at least one of the search terms values must match the query, but unlike 'phrase', the query only needs to match part of the value.", "enum": ["keywords", "text", "phrase"], "default": "keywords" }`
groups	`"groups": { "type": "string", "title": "Rule Group", "description": "Group for this rule." }`
effective_range	`"effective_range": { "type": "array", "title": "Effective Range", "description": "Time range when this rule is in effect.", "items": { "type": "string" } }`

Parameter	Schema
priority	"priority": { "type": "integer", "title": "Precedence", "description": "Used when multiple rules could take effect for a given query - higher number means higher precedence. For example, a rule with precedence of 10 will take effect before a rule with precedence of 3. Ties on precedence are broken based on creation time - the more recently created rule will take effect before the older rule." }
search_terms	"search_terms": { "type": "array", "title": "Search Terms", "description": "Search terms that trigger this rule.", "items": { "type": "string" } }
filters	"filters": { "type": "array", "title": "Filters", "description": "Filters used to determine if this rule triggers", "items": { "type": "object", "required": ["key", "value"], "properties": { "key": { "type": "string", "title": "Field Name" }, "value": { "type": "string", "title": "Field Value" } } } }
response_values	"response_values": { "type": "array", "title": "Set Response Values", "description": "Key/Value pairs to set in the fusion section of the response.", "items": { "type": "object", "required": ["key", "value"], "properties": { "key": { "type": "string", "title": "Field Name" }, "value": { "type": "string", "title": "Field Value" } } } }
query_profile_id_ss	"query_profile_id_ss": { "type": "array", "title": "Query Profile ID", "description": "Query Profile that triggers this rule", "items": { "type": "string" } }

Parameter	Schema
field_name Required	"field_name": { "type": "string", "title": "Field Name", "description": "Set the field name used by this rule to match the specified values.", "default": "id" }
field_values Required	"field_values": { "type": "array", "title": "Field Values", "description": "List of values for the specified field.", "minItems": 1, "items": { "type": "string" } }
use_qec	"use_qec": { "type": "boolean", "title": "Use Query Elevation Component", "description": "Set true to use Solr's QEC to force-elevate documents to the top. QEC only supports elevation through \"id\" field. Failed to config QEC in Solr correctly while having this box checked will cause this rule to stop working.", "default": false }
type Required	"type": { "type": "string", "title": "Type", "enum": ["boost_list"], "default": "boost_list", "hints": ["readonly"] }
doc_type	"doc_type": { "type": "string", "title": "Doc Type", "enum": ["rule"], "default": "rule", "hints": ["readonly"] }
display_type	"display_type": { "type": "string", "title": "Display Type", "enum": ["Boost List"], "default": "Boost List", "hints": ["readonly"] } }

Parameter	Schema
additionalProperties	
additionalProperties	`"additionalProperties": true`

Bury List Rule API Configurations

This rule type is used in the *Query Rewrite API*.

For a complete list of rule and rewrite types in the Query Rewrite API, see *Query Rewrite API Configurations*.

Bury List

The Bury List action buries documents with the selected values. This action minimizes certain documents without blocking them.

For example, a workplace may bury out-of-date documents.

General information

- Endpoint: `https://FUSION_HOST:FUSION_PORT/api/apps/APP_NAME/query-rewrite/schema/RULE_TYPE`
- Required parameters are:
  - name
  - field_name
  - field_values
  - type
- The values for the `hints` parameter include:
  - `hidden`: Do not display in the Fusion UI, but can be read and written in the schema.
  - `readonly`: Cannot be set by the user. If attempted, an error is generated and the operation fails.

Parameter	Schema
type	
type	string
properties	
id	`"id": { "type": "string", "title": "ID", "description": "The ID for this Rule. Used in the API to reference this query rewrite rule. Allowed characters: a-z, A-Z, 0-9, dash (-) and underscore (_)", "maxLength": 128, "pattern": "^[A-Za-z0-9_\\-]+$" }`
source	`"source": { "type": "string", "title": "Source", "description": "The source for this query rewrite rule, such as a Spark Job.", "hints": [ "hidden" ] }`
sourceType	`"sourceType": { "type": "string", "title": "Source Type", "description": "Source type", "hints": [ "hidden" ] }`
createdOn	`"createdOn": { "type": "string", "title": "Created Date/Time", "description": "Date/Time this rule was created.", "format": "date-time" }`
updatedOn	`"updatedOn": { "type": "string", "title": "Updated Date/Time", "description": "Date/Time this rule was last updated.", "format": "date-time" }`

Parameter	Schema
reviewer	`"reviewer": { "type": "string", "title": "Reviewer", "description": "Who reviewed this rewrite rule.", "hints": [ "hidden" ] }`
reviewedOn	`"reviewedOn": { "type": "string", "title": "Reviewed Date/Time", "description": "Date/Time this rule was last reviewed.", "hints": [ "hidden" ], "format": "date-time" }`
review	`"review": { "type": "string", "title": "Review", "enum": [ "approved", "denied" ], "default": "approved", "hints": [ "readonly" ] }`
deployed	`"deployed": { "type": "boolean", "title": "Deployed Flag", "description": "Flag to indicate if this rewrite rule has been deployed.", "hints": [ "hidden" ] }`
tags	`"tags": { "type": "array", "title": "Tags", "description": "Conditional tags to restrict the triggering of this query rewrite rule.", "items": { "type": "string" } }`
name Required	`"name": { "type": "string", "title": "Name", "description": "User-friendly name for this rule." }`

Parameter	Schema
description	`"description": { "type": "string", "title": "Description", "description": "Short description of what this rule does." }`
enabled	`"enabled": { "type": "boolean", "title": "Enabled", "description": "Set to false to disable this rule.", "default": true }`
matching	`"matching": { "type": "string", "title": "Match Query Using", "description": "Defines how the specified search terms will be compared to the user's query to determine whether this rule applies. 'keywords' means the query must match the keywords of the search terms value exactly. 'phrase' means the search query must contain one of the search terms values as a subphrase. 'text' means that at least one of the search terms values must match the query, but unlike 'phrase', the query only needs to match part of the value.", "enum": ["keywords", "text", "phrase"], "default": "keywords" }`
groups	`"groups": { "type": "string", "title": "Rule Group", "description": "Group for this rule." }`
effective_range	`"effective_range": { "type": "array", "title": "Effective Range", "description": "Time range when this rule is in effect.", "items": { "type": "string" } }`

Parameter	Schema
priority	"priority": { "type": "integer", "title": "Precedence", "description": "Used when multiple rules could take effect for a given query - higher number means higher precedence. For example, a rule with precedence of 10 will take effect before a rule with precedence of 3. Ties on precedence are broken based on creation time - the more recently created rule will take effect before the older rule." }
search_terms	"search_terms": { "type": "array", "title": "Search Terms", "description": "Search terms that trigger this rule.", "items": { "type": "string" } }
filters	"filters": { "type": "array", "title": "Filters", "description": "Filters used to determine if this rule triggers", "items": { "type": "object", "required": ["key", "value"], "properties": { "key": { "type": "string", "title": "Field Name" }, "value": { "type": "string", "title": "Field Value" } } } }
response_values	"response_values": { "type": "array", "title": "Set Response Values", "description": "Key/Value pairs to set in the fusion section of the response.", "items": { "type": "object", "required": ["key", "value"], "properties": { "key": { "type": "string", "title": "Field Name" }, "value": { "type": "string", "title": "Field Value" } } } }
query_profile_id_ss	"query_profile_id_ss": { "type": "array", "title": "Query Profile ID", "description": "Query Profile that triggers this rule", "items": { "type": "string" } }

Parameter	Schema
field_name Required	"field_name": { "type": "string", "title": "Field Name", "description": "Set the field name used by this rule to match the specified values.", "default": "id" }
field_values Required	"field_values": { "type": "array", "title": "Field Values", "description": "List of values for the specified field.", "minItems": 1, "items": { "type": "string" } }
type Required	"type": { "type": "string", "title": "Type", "enum": ["bury_list"], "default": "bury_list", "hints": ["readonly"] }
doc_type	"doc_type": { "type": "string", "title": "Doc Type", "enum": ["rule"], "default": "rule", "hints": ["readonly"] }
display_type	"display_type": { "type": "string", "title": "Display Type", "enum": ["Bury List"], "default": "Bury List", "hints": ["readonly"] } }
additionalProperties	
additionalProperties	"additionalProperties": true

Filter List Rule API Configurations

This rule type is used in the *Query Rewrite API*.

For a complete list of rule and rewrite types in the Query Rewrite API, see *Query Rewrite API Configurations*.

Filter List

The Filter List action applies filter queries *(https://solr.apache.org/guide/common-query-parameters.html)* (`fq`) to the query and changes the results so a pre-selected set of content displays.

For example, an e-commerce website may display only movies rated G or PG when a user searches for children's movies.

General information

- Endpoint: `https://FUSION_HOST:FUSION_PORT/api/apps/APP_NAME/query-rewrite/schema/RULE_TYPE`
- Required parameters are:
  - name
  - field_name
  - field_values
  - type
- The values for the `hints` parameter include:
  - `hidden`: Do not display in the Fusion UI, but can be read and written in the schema.
  - `readonly`: Cannot be set by the user. If attempted, an error is generated and the operation fails.

Parameter	Schema
type	
type	string
properties	
id	`"id": { "type": "string", "title": "ID", "description": "The ID for this Rule. Used in the API to reference this query rewrite rule. Allowed characters: a-z, A-Z, 0-9, dash (-) and underscore (_)", "maxLength": 128, "pattern": "^[A-Za-z0-9_\\-]+$" }`
source	`"source": { "type": "string", "title": "Source", "description": "The source for this query rewrite rule, such as a Spark Job.", "hints": [ "hidden" ] }`
sourceType	`"sourceType": { "type": "string", "title": "Source Type", "description": "Source type", "hints": [ "hidden" ] }`
createdOn	`"createdOn": { "type": "string", "title": "Created Date/Time", "description": "Date/Time this rule was created.", "format": "date-time" }`
updatedOn	`"updatedOn": { "type": "string", "title": "Updated Date/Time", "description": "Date/Time this rule was last updated.", "format": "date-time" }`

Parameter	Schema
reviewer	`"reviewer": { "type": "string", "title": "Reviewer", "description": "Who reviewed this rewrite rule.", "hints": [ "hidden" ] }`
reviewedOn	`"reviewedOn": { "type": "string", "title": "Reviewed Date/Time", "description": "Date/Time this rule was last reviewed.", "hints": [ "hidden" ], "format": "date-time" }`
review	`"review": { "type": "string", "title": "Review", "enum": [ "approved", "denied" ], "default": "approved", "hints": [ "readonly" ] }`
deployed	`"deployed": { "type": "boolean", "title": "Deployed Flag", "description": "Flag to indicate if this rewrite rule has been deployed.", "hints": [ "hidden" ] }`
tags	`"tags": { "type": "array", "title": "Tags", "description": "Conditional tags to restrict the triggering of this query rewrite rule.", "items": { "type": "string" } }`
name Required	`"name": { "type": "string", "title": "Name", "description": "User-friendly name for this rule." }`

Parameter	Schema
description	`"description": { "type": "string", "title": "Description", "description": "Short description of what this rule does." }`
enabled	`"enabled": { "type": "boolean", "title": "Enabled", "description": "Set to false to disable this rule.", "default": true }`
matching	`"matching": { "type": "string", "title": "Match Query Using", "description": "Defines how the specified search terms will be compared to the user's query to determine whether this rule applies. 'keywords' means the query must match the keywords of the search terms value exactly. 'phrase' means the search query must contain one of the search terms values as a subphrase. 'text' means that at least one of the search terms values must match the query, but unlike 'phrase', the query only needs to match part of the value.", "enum": ["keywords", "text", "phrase"], "default": "keywords" }`
groups	`"groups": { "type": "string", "title": "Rule Group", "description": "Group for this rule." }`
effective_range	`"effective_range": { "type": "array", "title": "Effective Range", "description": "Time range when this rule is in effect.", "items": { "type": "string" } }`

Parameter	Schema
priority	"priority": { "type": "integer", "title": "Precedence", "description": "Used when multiple rules could take effect for a given query - higher number means higher precedence. For example, a rule with precedence of 10 will take effect before a rule with precedence of 3. Ties on precedence are broken based on creation time - the more recently created rule will take effect before the older rule." }
search_terms	"search_terms": { "type": "array", "title": "Search Terms", "description": "Search terms that trigger this rule.", "items": { "type": "string" } }
filters	"filters": { "type": "array", "title": "Filters", "description": "Filters used to determine if this rule triggers", "items": { "type": "object", "required": ["key", "value"], "properties": { "key": { "type": "string", "title": "Field Name" }, "value": { "type": "string", "title": "Field Value" } } } }
response_values	"response_values": { "type": "array", "title": "Set Response Values", "description": "Key/Value pairs to set in the fusion section of the response.", "items": { "type": "object", "required": ["key", "value"], "properties": { "key": { "type": "string", "title": "Field Name" }, "value": { "type": "string", "title": "Field Value" } } } }
query_profile_id_ss	"query_profile_id_ss": { "type": "array", "title": "Query Profile ID", "description": "Query Profile that triggers this rule", "items": { "type": "string" } }

Parameter	Schema
field_name Required	`"field_name": { "type": "string", "title": "Field Name", "description": "Set the field name used by this rule to match the specified values.", "default": "id" }`
field_values Required	`"field_values": { "type": "array", "title": "Field Values", "description": "List of values for the specified field.", "minItems": 1, "items": { "type": "string" } }`
use_qec	`"use_qec": { "type": "boolean", "title": "Use Query Elevation Component", "description": "Set true to use Solr's QEC to force-elevate documents to the top. QEC only supports elevation through \"id\" field. Failed to config QEC in Solr correctly while having this box checked will cause this rule to stop working.", "default": false }`
terms_query_separator_s	`"terms_query_separator_s": { "type": "string", "title": "Terms Query Separator", "description": "Separator delimiting the values in the terms query string, defaulting to a comma.", "default": ",", "hints": [ "hidden" ] }`
terms_query_method_s	`"terms_query_method_s": { "type": "string", "title": "Terms Query Method", "description": "One of termsFilter (default), booleanQuery, automaton, docValuesTermsFilter", "default": "termsFilter", "hints": [ "hidden" ] }`

Parameter	Schema
type Required	"type": { "type": "string", "title": "Type", "enum": ["filter_list"], "default": "filter_list", "hints": ["readonly"] }
doc_type	"doc_type": { "type": "string", "title": "Doc Type", "enum": ["rule"], "default": "rule", "hints": ["readonly"] }
display_type	"display_type": { "type": "string", "title": "Display Type", "enum": ["Filter List"], "default": "Filter List", "hints": ["readonly"] } }
additionalProperties	
additionalProperties	"additionalProperties": true

Ingroup Boost List Rule API Configurations

This rule type is used in the *Query Rewrite API*.

For a complete list of rule and rewrite types in the Query Rewrite API, see *Query Rewrite API Configurations*.

Ingroup Boost List

The Ingroup Boost List action boosts an item within a product group. This action does not boost the product group.

General information

- Endpoint: `https://FUSION_HOST:FUSION_PORT/api/apps/APP_NAME/query-rewrite/schema/RULE_TYPE`
- Required parameters are:
  - name
  - field_name
  - field_values
  - groupField_s
  - type
- The values for the `hints` parameter include:
  - `hidden`: Do not display in the Fusion UI, but can be read and written in the schema.
  - `readonly`: Cannot be set by the user. If attempted, an error is generated and the operation fails.

Parameter	Schema
type	
type	string
properties	
id	`"id": { "type": "string", "title": "ID", "description": "The ID for this Rule. Used in the API to reference this query rewrite rule. Allowed characters: a-z, A-Z, 0-9, dash (-) and underscore (_)", "maxLength": 128, "pattern": "^[A-Za-z0-9_\\-]+$" }`
source	`"source": { "type": "string", "title": "Source", "description": "The source for this query rewrite rule, such as a Spark Job.", "hints": [ "hidden" ] }`
sourceType	`"sourceType": { "type": "string", "title": "Source Type", "description": "Source type", "hints": [ "hidden" ] }`
createdOn	`"createdOn": { "type": "string", "title": "Created Date/Time", "description": "Date/Time this rule was created.", "format": "date-time" }`
updatedOn	`"updatedOn": { "type": "string", "title": "Updated Date/Time", "description": "Date/Time this rule was last updated.", "format": "date-time" }`

Parameter	Schema
reviewer	`"reviewer": { "type": "string", "title": "Reviewer", "description": "Who reviewed this rewrite rule.", "hints": [ "hidden" ] }`
reviewedOn	`"reviewedOn": { "type": "string", "title": "Reviewed Date/Time", "description": "Date/Time this rule was last reviewed.", "hints": [ "hidden" ], "format": "date-time" }`
review	`"review": { "type": "string", "title": "Review", "enum": [ "approved", "denied" ], "default": "approved", "hints": [ "readonly" ] }`
deployed	`"deployed": { "type": "boolean", "title": "Deployed Flag", "description": "Flag to indicate if this rewrite rule has been deployed.", "hints": [ "hidden" ] }`
tags	`"tags": { "type": "array", "title": "Tags", "description": "Conditional tags to restrict the triggering of this query rewrite rule.", "items": { "type": "string" } }`
name Required	`"name": { "type": "string", "title": "Name", "description": "User-friendly name for this rule." },`

Parameter	Schema
description	`"description": { "type": "string", "title": "Description", "description": "Short description of what this rule does." }`
enabled	`"enabled": { "type": "boolean", "title": "Enabled", "description": "Set to false to disable this rule.", "default": true }`
matching	`"matching": { "type": "string", "title": "Match Query Using", "description": "Defines how the specified search terms will be compared to the user's query to determine whether this rule applies. 'keywords' means the query must match the keywords of the search terms value exactly. 'phrase' means the search query must contain one of the search terms values as a subphrase. 'text' means that at least one of the search terms values must match the query, but unlike 'phrase', the query only needs to match part of the value.", "enum": ["keywords", "text", "phrase"], "default": "keywords" }`
groups	`"groups": { "type": "string", "title": "Rule Group", "description": "Group for this rule." }`
effective_range	`"effective_range": { "type": "array", "title": "Effective Range", "description": "Time range when this rule is in effect.", "items": { "type": "string" } }`

Parameter	Schema
priority	"priority": { "type": "integer", "title": "Precedence", "description": "Used when multiple rules could take effect for a given query - higher number means higher precedence. For example, a rule with precedence of 10 will take effect before a rule with precedence of 3. Ties on precedence are broken based on creation time - the more recently created rule will take effect before the older rule." }
search_terms	"search_terms": { "type": "array", "title": "Search Terms", "description": "Search terms that trigger this rule.", "items": { "type": "string" } }
filters	"filters": { "type": "array", "title": "Filters", "description": "Filters used to determine if this rule triggers", "items": { "type": "object", "required": ["key", "value"], "properties": { "key": { "type": "string", "title": "Field Name" }, "value": { "type": "string", "title": "Field Value" } } } }
response_values	"response_values": { "type": "array", "title": "Set Response Values", "description": "Key/Value pairs to set in the fusion section of the response.", "items": { "type": "object", "required": ["key", "value"], "properties": { "key": { "type": "string", "title": "Field Name" }, "value": { "type": "string", "title": "Field Value" } } } }
query_profile_id_ss	"query_profile_id_ss": { "type": "array", "title": "Query Profile ID", "description": "Query Profile that triggers this rule", "items": { "type": "string" } }

Parameter	Schema
field_name Required	"field_name": { "type": "string", "title": "Field Name", "description": "Set the field name used by this rule to match the specified values.", "default": "id" }
field_values Required	"field_values": { "type": "array", "title": "Field Values", "description": "List of values for the specified field.", "minItems": 1, "items": { "type": "string" } }
groupField_s Required	"groupField_s": { "type": "string", "title": "Grouping Field", "description": "A single field name that must be grouped on for this rule to be applied. If the request does not have a 'group.field' parameter matching this property, the rule is not applied." }
type Required	"type": { "type": "string", "title": "Type", "enum": ["ingroup_boost_list"], "default": "ingroup_boost_list", "hints": ["readonly"] }
doc_type	"doc_type": { "type": "string", "title": "Doc Type", "enum": ["rule"], "default": "rule", "hints": ["readonly"] }
display_type	"display_type": { "type": "string", "title": "Display Type", "enum": ["Ingroup Boost List"], "default": "Ingroup Boost List", "hints": ["readonly"] } }

Parameter	Schema
additionalProperties	
additionalProperties	`"additionalProperties": true`

Ingroup Bury List Rule API Configurations

This rule type is used in the *Query Rewrite API*.

For a complete list of rule and rewrite types in the Query Rewrite API, see *Query Rewrite API Configurations*.

Ingroup Bury List

The Ingroup Bury List action buries an item within a product group. This action does not bury the product group.

General information

- Endpoint: `https://FUSION_HOST:FUSION_PORT/api/apps/APP_NAME/query-rewrite/schema/RULE_TYPE`
- Required parameters are:
  - name
  - field_name
  - field_values
  - groupField_s
  - type
- The values for the `hints` parameter include:
  - `hidden`: Do not display in the Fusion UI, but can be read and written in the schema.
  - `readonly`: Cannot be set by the user. If attempted, an error is generated and the operation fails.

Parameter	Schema
type	
type	string
properties	
id	`"id": { "type": "string", "title": "ID", "description": "The ID for this Rule. Used in the API to reference this query rewrite rule. Allowed characters: a-z, A-Z, 0-9, dash (-) and underscore (_)", "maxLength": 128, "pattern": "^[A-Za-z0-9_\\-]+$" }`
source	`"source": { "type": "string", "title": "Source", "description": "The source for this query rewrite rule, such as a Spark Job.", "hints": [ "hidden" ] }`
sourceType	`"sourceType": { "type": "string", "title": "Source Type", "description": "Source type", "hints": [ "hidden" ] }`
createdOn	`"createdOn": { "type": "string", "title": "Created Date/Time", "description": "Date/Time this rule was created.", "format": "date-time" }`
updatedOn	`"updatedOn": { "type": "string", "title": "Updated Date/Time", "description": "Date/Time this rule was last updated.", "format": "date-time" }`

Parameter	Schema
reviewer	`"reviewer": { "type": "string", "title": "Reviewer", "description": "Who reviewed this rewrite rule.", "hints": [ "hidden" ] }`
reviewedOn	`"reviewedOn": { "type": "string", "title": "Reviewed Date/Time", "description": "Date/Time this rule was last reviewed.", "hints": [ "hidden" ], "format": "date-time" }`
review	`"review": { "type": "string", "title": "Review", "enum": [ "approved", "denied" ], "default": "approved", "hints": [ "readonly" ] }`
deployed	`"deployed": { "type": "boolean", "title": "Deployed Flag", "description": "Flag to indicate if this rewrite rule has been deployed.", "hints": [ "hidden" ] }`
tags	`"tags": { "type": "array", "title": "Tags", "description": "Conditional tags to restrict the triggering of this query rewrite rule.", "items": { "type": "string" } }`
name Required	`"name": { "type": "string", "title": "Name", "description": "User-friendly name for this rule." }`

Parameter	Schema
description	`"description": { "type": "string", "title": "Description", "description": "Short description of what this rule does." }`
enabled	`"enabled": { "type": "boolean", "title": "Enabled", "description": "Set to false to disable this rule.", "default": true }`
matching	`"matching": { "type": "string", "title": "Match Query Using", "description": "Defines how the specified search terms will be compared to the user's query to determine whether this rule applies. 'keywords' means the query must match the keywords of the search terms value exactly. 'phrase' means the search query must contain one of the search terms values as a subphrase. 'text' means that at least one of the search terms values must match the query, but unlike 'phrase', the query only needs to match part of the value.", "enum": ["keywords", "text", "phrase"], "default": "keywords" }`
groups	`"groups": { "type": "string", "title": "Rule Group", "description": "Group for this rule." }`
effective_range	`"effective_range": { "type": "array", "title": "Effective Range", "description": "Time range when this rule is in effect.", "items": { "type": "string" } }`

Parameter	Schema
priority	"priority": { "type": "integer", "title": "Precedence", "description": "Used when multiple rules could take effect for a given query - higher number means higher precedence. For example, a rule with precedence of 10 will take effect before a rule with precedence of 3. Ties on precedence are broken based on creation time - the more recently created rule will take effect before the older rule." }
search_terms	"search_terms": { "type": "array", "title": "Search Terms", "description": "Search terms that trigger this rule.", "items": { "type": "string" } }
filters	"filters": { "type": "array", "title": "Filters", "description": "Filters used to determine if this rule triggers", "items": { "type": "object", "required": ["key", "value"], "properties": { "key": { "type": "string", "title": "Field Name" }, "value": { "type": "string", "title": "Field Value" } } } }
response_values	"response_values": { "type": "array", "title": "Set Response Values", "description": "Key/Value pairs to set in the fusion section of the response.", "items": { "type": "object", "required": ["key", "value"], "properties": { "key": { "type": "string", "title": "Field Name" }, "value": { "type": "string", "title": "Field Value" } } } }
query_profile_id_ss	"query_profile_id_ss": { "type": "array", "title": "Query Profile ID", "description": "Query Profile that triggers this rule", "items": { "type": "string" } }

Parameter	Schema
field_name Required	"field_name": { "type": "string", "title": "Field Name", "description": "Set the field name used by this rule to match the specified values.", "default": "id" }
field_values Required	"field_values": { "type": "array", "title": "Field Values", "description": "List of values for the specified field.", "minItems": 1, "items": { "type": "string" } }
groupField_s Required	"groupField_s": { "type": "string", "title": "Grouping Field", "description": "A single field name that must be grouped on for this rule to be applied. If the request does not have a 'group.field' parameter matching this property, the rule is not applied." }
type Required	"type": { "type": "string", "title": "Type", "enum": ["ingroup_bury_list"], "default": "ingroup_bury_list", "hints": ["readonly"] }
doc_type	"doc_type": { "type": "string", "title": "Doc Type", "enum": ["rule"], "default": "rule", "hints": ["readonly"] }
display_type	"display_type": { "type": "string", "title": "Display Type", "enum": ["Ingroup Bury List"], "default": "Ingroup Bury List", "hints": ["readonly"] } }

Parameter	Schema
additionalProperties	
additionalProperties	`"additionalProperties": true`

Ingroup Pinned List Rule API Configurations

This rule type is used in the *Query Rewrite API*.

For a complete list of rule and rewrite types in the Query Rewrite API, see *Query Rewrite API Configurations*.

Ingroup Pinned List

The Ingroup Pinned action sticks a document to a specific location within a product group. This ensures that users view a result in a specified place. This action does not pin the product group.

General information

- Endpoint: `https://FUSION_HOST:FUSION_PORT/api/apps/APP_NAME/query-rewrite/schema/RULE_TYPE`
- Required parameters are:
  - name
  - field_name
  - groupField_s
  - groupValue_s
  - pinned_values
  - type
- The values for the `hints` parameter include:
  - `hidden`: Do not display in the Fusion UI, but can be read and written in the schema.
  - `readonly`: Cannot be set by the user. If attempted, an error is generated and the operation fails.

Parameter	Schema
type	
type	string
properties	
id	`"id": { "type": "string", "title": "ID", "description": "The ID for this Rule. Used in the API to reference this query rewrite rule. Allowed characters: a-z, A-Z, 0-9, dash (-) and underscore (_)", "maxLength": 128, "pattern": "^[A-Za-z0-9_\\-]+$" }`
source	`"source": { "type": "string", "title": "Source", "description": "The source for this query rewrite rule, such as a Spark Job.", "hints": [ "hidden" ] }`
sourceType	`"sourceType": { "type": "string", "title": "Source Type", "description": "Source type", "hints": [ "hidden" ] }`
createdOn	`"createdOn": { "type": "string", "title": "Created Date/Time", "description": "Date/Time this rule was created.", "format": "date-time" }`
updatedOn	`"updatedOn": { "type": "string", "title": "Updated Date/Time", "description": "Date/Time this rule was last updated.", "format": "date-time" }`

Parameter	Schema
reviewer	`"reviewer": { "type": "string", "title": "Reviewer", "description": "Who reviewed this rewrite rule.", "hints": [ "hidden" ] }`
reviewedOn	`"reviewedOn": { "type": "string", "title": "Reviewed Date/Time", "description": "Date/Time this rule was last reviewed.", "hints": [ "hidden" ], "format": "date-time" }`
review	`"review": { "type": "string", "title": "Review", "enum": [ "approved", "denied" ], "default": "approved", "hints": [ "readonly" ] }`
deployed	`"deployed": { "type": "boolean", "title": "Deployed Flag", "description": "Flag to indicate if this rewrite rule has been deployed.", "hints": [ "hidden" ] }`
tags	`"tags": { "type": "array", "title": "Tags", "description": "Conditional tags to restrict the triggering of this query rewrite rule.", "items": { "type": "string" } }`
name Required	`"name": { "type": "string", "title": "Name", "description": "User-friendly name for this rule." }`

Parameter	Schema
description	`"description": { "type": "string", "title": "Description", "description": "Short description of what this rule does." }`
enabled	`"enabled": { "type": "boolean", "title": "Enabled", "description": "Set to false to disable this rule.", "default": true }`
matching	`"matching": { "type": "string", "title": "Match Query Using", "description": "Defines how the specified search terms will be compared to the user's query to determine whether this rule applies. 'keywords' means the query must match the keywords of the search terms value exactly. 'phrase' means the search query must contain one of the search terms values as a subphrase. 'text' means that at least one of the search terms values must match the query, but unlike 'phrase', the query only needs to match part of the value.", "enum": ["keywords", "text", "phrase"], "default": "keywords" }`
groups	`"groups": { "type": "string", "title": "Rule Group", "description": "Group for this rule." }`
effective_range	`"effective_range": { "type": "array", "title": "Effective Range", "description": "Time range when this rule is in effect.", "items": { "type": "string" } }`

Parameter	Schema
priority	"priority": { "type": "integer", "title": "Precedence", "description": "Used when multiple rules could take effect for a given query - higher number means higher precedence. For example, a rule with precedence of 10 will take effect before a rule with precedence of 3. Ties on precedence are broken based on creation time - the more recently created rule will take effect before the older rule." }
search_terms	"search_terms": { "type": "array", "title": "Search Terms", "description": "Search terms that trigger this rule.", "items": { "type": "string" } }
filters	"filters": { "type": "array", "title": "Filters", "description": "Filters used to determine if this rule triggers", "items": { "type": "object", "required": ["key", "value"], "properties": { "key": { "type": "string", "title": "Field Name" }, "value": { "type": "string", "title": "Field Value" } } } }
response_values	"response_values": { "type": "array", "title": "Set Response Values", "description": "Key/Value pairs to set in the fusion section of the response.", "items": { "type": "object", "required": ["key", "value"], "properties": { "key": { "type": "string", "title": "Field Name" }, "value": { "type": "string", "title": "Field Value" } } } }
query_profile_id_ss	"query_profile_id_ss": { "type": "array", "title": "Query Profile ID", "description": "Query Profile that triggers this rule", "items": { "type": "string" } }

Parameter	Schema
field_name Required	"field_name": { "type": "string", "title": "Field Name", "description": "Set the field name used by this rule to match the specified values.", "default": "id" }
field_values	"field_values": { "type": "array", "title": "Field Values", "description": "List of values for the specified field.", "hints": ["hidden"], "minItems": 1, "items": { "type": "string" } }
groupField_s Required	"groupField_s": { "type": "string", "title": "Grouping Field", "description": "A single field name that must be grouped on for this rule to be applied. If the request does not have a 'group.field' parameter matching this property, the rule is not applied." }
groupValue_s Required	"groupValue_s": { "type": "string", "title": "Pin Within A Specific Group Value", "description": "A single group value in which the pin rule will apply to the docs." }
pinned_values Required	"pinned_values": { "type": "array", "title": "Pinned Documents", "description": "A list of one or more documents to boost and pin to a specific position in the group (1-based index)", "minItems": 1, "items": { "type": "object", "required": ["value", "position"], "properties": { "value": { "type": "string", "title": "Value", "description": "Value to boost" }, "position": { "type": "integer", "title": "Position", "description": "Absolute position in the first page of results (1-based index)" } } } }

Parameter	Schema
type Required	`"type": { "type": "string", "title": "Type", "enum": [ "ingroup_pinned" ], "default": "ingroup_pinned", "hints": [ "readonly" ] }`
doc_type	`"doc_type": { "type": "string", "title": "Doc Type", "enum": [ "rule" ], "default": "rule", "hints": [ "readonly" ] }`
display_type	`"display_type": { "type": "string", "title": "Display Type", "enum": [ "Ingroup Pinned" ], "default": "Ingroup Pinned", "hints": [ "readonly" ] } }`
additionalProperties	
additionalProperties	`"additionalProperties": true`

JSON Blob Rule API Configurations

This rule type is used in the *Query Rewrite API*.

For a complete list of rule and rewrite types in the Query Rewrite API, see *Query Rewrite API Configurations*.

JSON Blob

The JSON Blob action returns an *arbitrary JSON blob* to your frontend when a business rule fires. The JSON blob displays outside of the search results.

General information

- Endpoint: `https://FUSION_HOST:FUSION_PORT/api/apps/APP_NAME/query-rewrite/schema/RULE_TYPE`
- Required parameters are:
  - name
  - json
  - type
- The values for the `hints` parameter include:
  - `hidden`: Do not display in the Fusion UI, but can be read and written in the schema.
  - `readonly`: Cannot be set by the user. If attempted, an error is generated and the operation fails.

Parameter	Schema
type	
type	string
properties	
id	`"id": { "type": "string", "title": "ID", "description": "The ID for this Rule. Used in the API to reference this query rewrite rule. Allowed characters: a-z, A-Z, 0-9, dash (-) and underscore (_)", "maxLength": 128, "pattern": "^[A-Za-z0-9_\\-]+$" }`
source	`"source": { "type": "string", "title": "Source", "description": "The source for this query rewrite rule, such as a Spark Job.", "hints": [ "hidden" ] }`
sourceType	`"sourceType": { "type": "string", "title": "Source Type", "description": "Source type", "hints": [ "hidden" ] }`
createdOn	`"createdOn": { "type": "string", "title": "Created Date/Time", "description": "Date/Time this rule was created.", "format": "date-time" }`
updatedOn	`"updatedOn": { "type": "string", "title": "Updated Date/Time", "description": "Date/Time this rule was last updated.", "format": "date-time" }`

Parameter	Schema
reviewer	`"reviewer": { "type": "string", "title": "Reviewer", "description": "Who reviewed this rewrite rule.", "hints": [ "hidden" ] }`
reviewedOn	`"reviewedOn": { "type": "string", "title": "Reviewed Date/Time", "description": "Date/Time this rule was last reviewed.", "hints": [ "hidden" ], "format": "date-time" }`
review	`"review": { "type": "string", "title": "Review", "enum": [ "approved", "denied" ], "default": "approved", "hints": [ "readonly" ] }`
deployed	`"deployed": { "type": "boolean", "title": "Deployed Flag", "description": "Flag to indicate if this rewrite rule has been deployed.", "hints": [ "hidden" ] }`
tags	`"tags": { "type": "array", "title": "Tags", "description": "Conditional tags to restrict the triggering of this query rewrite rule.", "items": { "type": "string" } }`
name Required	`"name": { "type": "string", "title": "Name", "description": "User-friendly name for this rule." }`

Parameter	Schema
description	`"description": { "type": "string", "title": "Description", "description": "Short description of what this rule does." }`
enabled	`"enabled": { "type": "boolean", "title": "Enabled", "description": "Set to false to disable this rule.", "default": true }`
matching	`"matching": { "type": "string", "title": "Match Query Using", "description": "Defines how the specified search terms will be compared to the user's query to determine whether this rule applies. 'keywords' means the query must match the keywords of the search terms value exactly. 'phrase' means the search query must contain one of the search terms values as a subphrase. 'text' means that at least one of the search terms values must match the query, but unlike 'phrase', the query only needs to match part of the value.", "enum": ["keywords", "text", "phrase"], "default": "keywords" }`
groups	`"groups": { "type": "string", "title": "Rule Group", "description": "Group for this rule." }`
effective_range	`"effective_range": { "type": "array", "title": "Effective Range", "description": "Time range when this rule is in effect.", "items": { "type": "string" } }`

Parameter	Schema
priority	"priority": { "type": "integer", "title": "Precedence", "description": "Used when multiple rules could take effect for a given query - higher number means higher precedence. For example, a rule with precedence of 10 will take effect before a rule with precedence of 3. Ties on precedence are broken based on creation time - the more recently created rule will take effect before the older rule." }
search_terms	"search_terms": { "type": "array", "title": "Search Terms", "description": "Search terms that trigger this rule.", "items": { "type": "string" } }
filters	"filters": { "type": "array", "title": "Filters", "description": "Filters used to determine if this rule triggers", "items": { "type": "object", "required": ["key", "value"], "properties": { "key": { "type": "string", "title": "Field Name" }, "value": { "type": "string", "title": "Field Value" } } } }
response_values	"response_values": { "type": "array", "title": "Set Response Values", "description": "Key/Value pairs to set in the fusion section of the response.", "hints": ["hidden"], "items": { "type": "object", "required": ["key", "value"], "properties": { "key": { "type": "string", "title": "Field Name" }, "value": { "type": "string", "title": "Field Value" } } } }
query_profile_id_ss	"query_profile_id_ss": { "type": "array", "title": "Query Profile ID", "description": "Query Profile that triggers this rule", "items": { "type": "string" } }

Parameter	Schema
blobType	"blobType": { "type": "string", "title": "Blob Type", "description": "Indicates what kind of data this blob contains. This only has an effect if your frontend has been developed to utilize this field. Thus, you will need to know what the possible values are and what effect they have on your frontend. This controls where the blob shows up in the Fusion response when triggered in a query - /fusion/jsonBlob/(blobtype). We recommended to use this field if you have different kinds of JSON Blobs, so your frontend can more easily and safely handle them. If no blob type is specified, defaults to 'default'.", "default": "default" }
json Required	"json": { "type": "string", "title": "JSON Blob", "description": "Arbitrary JSON blob" }
type Required	"type": { "type": "string", "title": "Type", "enum": ["json_blob"], "default": "json_blob", "hints": ["readonly"] }
doc_type	"doc_type": { "type": "string", "title": "Doc Type", "enum": ["rule"], "default": "rule", "hints": ["readonly"] }
display_type	"display_type": { "type": "string", "title": "Display Type", "enum": ["Json Blob"], "default": "Json Blob", "hints": ["readonly"] } }
additionalProperties	

Parameter	Schema
additionalProperties	"additionalProperties": true

Pinned Rule API Configurations

This rule type is used in the *Query Rewrite API*.

For a complete list of rule and rewrite types in the Query Rewrite API, see *Query Rewrite API Configurations*.

Pinned

The Pinned action sticks a document to a specific location within the search results. This ensures that users view a result in a specified place.

For example, a workplace may pin documents answering frequently asked questions.

General information

- Endpoint: `https://FUSION_HOST:FUSION_PORT/api/apps/APP_NAME/query-rewrite/schema/RULE_TYPE`
- Required parameters are:
  - name
  - field_name
  - pinned_values
  - type
- The values for the `hints` parameter include:
  - `hidden`: Do not display in the Fusion UI, but can be read and written in the schema.
  - `readonly`: Cannot be set by the user. If attempted, an error is generated and the operation fails.

Parameter	Schema
type	
type	string
properties	
id	`"id": { "type": "string", "title": "ID", "description": "The ID for this Rule. Used in the API to reference this query rewrite rule. Allowed characters: a-z, A-Z, 0-9, dash (-) and underscore (_)", "maxLength": 128, "pattern": "^[A-Za-z0-9_\\-]+$" }`
source	`"source": { "type": "string", "title": "Source", "description": "The source for this query rewrite rule, such as a Spark Job.", "hints": [ "hidden" ] }`
sourceType	`"sourceType": { "type": "string", "title": "Source Type", "description": "Source type", "hints": [ "hidden" ] }`
createdOn	`"createdOn": { "type": "string", "title": "Created Date/Time", "description": "Date/Time this rule was created.", "format": "date-time" }`
updatedOn	`"updatedOn": { "type": "string", "title": "Updated Date/Time", "description": "Date/Time this rule was last updated.", "format": "date-time" }`

Parameter	Schema
reviewer	`"reviewer": { "type": "string", "title": "Reviewer", "description": "Who reviewed this rewrite rule.", "hints": [ "hidden" ] }`
reviewedOn	`"reviewedOn": { "type": "string", "title": "Reviewed Date/Time", "description": "Date/Time this rule was last reviewed.", "hints": [ "hidden" ], "format": "date-time" }`
review	`"review": { "type": "string", "title": "Review", "enum": [ "approved", "denied" ], "default": "approved", "hints": [ "readonly" ] }`
deployed	`"deployed": { "type": "boolean", "title": "Deployed Flag", "description": "Flag to indicate if this rewrite rule has been deployed.", "hints": [ "hidden" ] }`
tags	`"tags": { "type": "array", "title": "Tags", "description": "Conditional tags to restrict the triggering of this query rewrite rule.", "items": { "type": "string" } }`
name Required	`"name": { "type": "string", "title": "Name", "description": "User-friendly name for this rule." }`

Parameter	Schema
description	`"description": { "type": "string", "title": "Description", "description": "Short description of what this rule does." }`
enabled	`"enabled": { "type": "boolean", "title": "Enabled", "description": "Set to false to disable this rule.", "default": true }`
matching	`"matching": { "type": "string", "title": "Match Query Using", "description": "Defines how the specified search terms will be compared to the user's query to determine whether this rule applies. 'keywords' means the query must match the keywords of the search terms value exactly. 'phrase' means the search query must contain one of the search terms values as a subphrase. 'text' means that at least one of the search terms values must match the query, but unlike 'phrase', the query only needs to match part of the value.", "enum": ["keywords", "text", "phrase"], "default": "keywords" }`
groups	`"groups": { "type": "string", "title": "Rule Group", "description": "Group for this rule." }`
effective_range	`"effective_range": { "type": "array", "title": "Effective Range", "description": "Time range when this rule is in effect.", "items": { "type": "string" } }`

Parameter	Schema
priority	"priority": { "type": "integer", "title": "Precedence", "description": "Used when multiple rules could take effect for a given query - higher number means higher precedence. For example, a rule with precedence of 10 will take effect before a rule with precedence of 3. Ties on precedence are broken based on creation time - the more recently created rule will take effect before the older rule." }
search_terms	"search_terms": { "type": "array", "title": "Search Terms", "description": "Search terms that trigger this rule.", "items": { "type": "string" } }
filters	"filters": { "type": "array", "title": "Filters", "description": "Filters used to determine if this rule triggers", "items": { "type": "object", "required": ["key", "value"], "properties": { "key": { "type": "string", "title": "Field Name" }, "value": { "type": "string", "title": "Field Value" } } } }
response_values	"response_values": { "type": "array", "title": "Set Response Values", "description": "Key/Value pairs to set in the fusion section of the response.", "items": { "type": "object", "required": ["key", "value"], "properties": { "key": { "type": "string", "title": "Field Name" }, "value": { "type": "string", "title": "Field Value" } } } }
query_profile_id_ss	"query_profile_id_ss": { "type": "array", "title": "Query Profile ID", "description": "Query Profile that triggers this rule", "items": { "type": "string" } }

Parameter	Schema
field_name Required	"field_name": { "type": "string", "title": "Field Name", "description": "Set the field name used by this rule to match the specified values.", "default": "id" }
field_values	"field_values": { "type": "array", "title": "Field Values", "description": "List of values for the specified field.", "hints": ["hidden"], "minItems": 1, "items": { "type": "string" } }
use_qec	"use_qec": { "type": "boolean", "title": "Use Query Elevation Component", "description": "Set true to use Solr's QEC to force-elevate documents to the top. QEC only supports elevation through \"id\" field. Failed to config QEC in Solr correctly while having this box checked will cause this rule to stop working.", "default": false }
pinned_values Required	"pinned_values": { "type": "array", "title": "Pinned Documents", "description": "A list of one or more documents to boost and pin to specific positions in the results (1-based index). Note that pinning will only be applied to the first page of results.", "minItems": 1, "items": { "type": "object", "required": ["value", "position"], "properties": { "value": { "type": "string", "title": "Value", "description": "Value to boost" }, "position": { "type": "integer", "title": "Position", "description": "Absolute position in the first page of results (1-based index)" } } } }
type Required	"type": { "type": "string", "title": "Type", "enum": ["pinned"], "default": "pinned", "hints": ["readonly"] }

Parameter	Schema
doc_type	"doc_type": { "type": "string", "title": "Doc Type", "enum": ["rule"], "default": "rule", "hints": ["readonly"] }
display_type	"display_type": { "type": "string", "title": "Display Type", "enum": ["Pinned"], "default": "Pinned", "hints": ["readonly"] } }
additionalProperties	
additionalProperties	"additionalProperties": true

Redirect Rule API Configurations

This rule type is used in the *Query Rewrite API*.

For a complete list of rule and rewrite types in the Query Rewrite API, see *Query Rewrite API Configurations*.

Redirect

The Redirect action displays a message at the top of the search results that the frontend can utilize to redirect a user to a specific URL. This action guides users to the desired URL when the search query isn't included in the collection.

For example, a workplace may create a redirect action that displays a link to the internal help desk when an employee searches for "computer broken".

General information

- Endpoint: `https://FUSION_HOST:FUSION_PORT/api/apps/APP_NAME/query-rewrite/schema/RULE_TYPE`
- Required parameters are:
  - name
  - redirect_url
  - type
- The values for the `hints` parameter include:
  - `hidden`: Do not display in the Fusion UI, but can be read and written in the schema.
  - `readonly`: Cannot be set by the user. If attempted, an error is generated and the operation fails.

Parameter	Schema
type	
type	string
properties	
id	`"id": { "type": "string", "title": "ID", "description": "The ID for this Rule. Used in the API to reference this query rewrite rule. Allowed characters: a-z, A-Z, 0-9, dash (-) and underscore (_)", "maxLength": 128, "pattern": "^[A-Za-z0-9_\\-]+$" }`
source	`"source": { "type": "string", "title": "Source", "description": "The source for this query rewrite rule, such as a Spark Job.", "hints": [ "hidden" ] }`
sourceType	`"sourceType": { "type": "string", "title": "Source Type", "description": "Source type", "hints": [ "hidden" ] }`
createdOn	`"createdOn": { "type": "string", "title": "Created Date/Time", "description": "Date/Time this rule was created.", "format": "date-time" }`
updatedOn	`"updatedOn": { "type": "string", "title": "Updated Date/Time", "description": "Date/Time this rule was last updated.", "format": "date-time" }`

Parameter	Schema
reviewer	`"reviewer": { "type": "string", "title": "Reviewer", "description": "Who reviewed this rewrite rule.", "hints": [ "hidden" ] }`
reviewedOn	`"reviewedOn": { "type": "string", "title": "Reviewed Date/Time", "description": "Date/Time this rule was last reviewed.", "hints": [ "hidden" ], "format": "date-time" }`
review	`"review": { "type": "string", "title": "Review", "enum": [ "approved", "denied" ], "default": "approved", "hints": [ "readonly" ] }`
deployed	`"deployed": { "type": "boolean", "title": "Deployed Flag", "description": "Flag to indicate if this rewrite rule has been deployed.", "hints": [ "hidden" ] }`
tags	`"tags": { "type": "array", "title": "Tags", "description": "Conditional tags to restrict the triggering of this query rewrite rule.", "items": { "type": "string" } }`
name Required	`"name": { "type": "string", "title": "Name", "description": "User-friendly name for this rule." }`

Parameter	Schema
description	`"description": { "type": "string", "title": "Description", "description": "Short description of what this rule does." }`
enabled	`"enabled": { "type": "boolean", "title": "Enabled", "description": "Set to false to disable this rule.", "default": true }`
matching	`"matching": { "type": "string", "title": "Match Query Using", "description": "Defines how the specified search terms will be compared to the user's query to determine whether this rule applies. 'keywords' means the query must match the keywords of the search terms value exactly. 'phrase' means the search query must contain one of the search terms values as a subphrase. 'text' means that at least one of the search terms values must match the query, but unlike 'phrase', the query only needs to match part of the value.", "enum": ["keywords", "text", "phrase"], "default": "keywords" }`
groups	`"groups": { "type": "string", "title": "Rule Group", "description": "Group for this rule." }`
effective_range	`"effective_range": { "type": "array", "title": "Effective Range", "description": "Time range when this rule is in effect.", "items": { "type": "string" } }`

Parameter	Schema
priority	"priority": { "type": "integer", "title": "Precedence", "description": "Used when multiple rules could take effect for a given query - higher number means higher precedence. For example, a rule with precedence of 10 will take effect before a rule with precedence of 3. Ties on precedence are broken based on creation time - the more recently created rule will take effect before the older rule." }
search_terms	"search_terms": { "type": "array", "title": "Search Terms", "description": "Search terms that trigger this rule.", "items": { "type": "string" } }
filters	"filters": { "type": "array", "title": "Filters", "description": "Filters used to determine if this rule triggers", "items": { "type": "object", "required": ["key", "value"], "properties": { "key": { "type": "string", "title": "Field Name" }, "value": { "type": "string", "title": "Field Value" } } } }
response_values	"response_values": { "type": "array", "title": "Set Response Values", "description": "Key/Value pairs to set in the fusion section of the response.", "hints": ["hidden"], "items": { "type": "object", "required": ["key", "value"], "properties": { "key": { "type": "string", "title": "Field Name" }, "value": { "type": "string", "title": "Field Value" } } } }
query_profile_id_ss	"query_profile_id_ss": { "type": "array", "title": "Query Profile ID", "description": "Query Profile that triggers this rule", "items": { "type": "string" } }

Parameter	Schema
redirect_url Required	`"redirect_url": { "type": "string", "title": "Redirect", "description": "Path or URL to redirect the response to" }`
type Required	`"type": { "type": "string", "title": "Type", "enum": [ "redirect" ], "default": "redirect", "hints": [ "readonly" ] }`
doc_type	`"doc_type": { "type": "string", "title": "Doc Type", "enum": [ "rule" ], "default": "rule", "hints": [ "readonly" ] }`
display_type	`"display_type": { "type": "string", "title": "Display Type", "enum": [ "Redirect" ], "default": "Redirect", "hints": [ "readonly" ] } }`
additionalProperties	
additionalProperties	`"additionalProperties": true`

Response Value Rule API Configurations

This rule type is used in the *Query Rewrite API*.

For a complete list of rule and rewrite types in the Query Rewrite API, see *Query Rewrite API Configurations*.

Response Value

The Response Value action sends an arbitrary value to the frontend or pipeline, which the frontend of pipeline can use to perform another action. The Response Value action can be combined with other actions within the same rule.

For example, an e-commerce website may use the response value action to display an advertisement.

General information

- Endpoint: `https://FUSION_HOST:FUSION_PORT/api/apps/APP_NAME/query-rewrite/schema/RULE_TYPE`
- Required parameters are:
  - name
  - type
- The values for the `hints` parameter include:
  - `hidden`: Do not display in the Fusion UI, but can be read and written in the schema.
  - `readonly`: Cannot be set by the user. If attempted, an error is generated and the operation fails.

Parameter	Schema
type	
type	string
properties	
id	`"id": { "type": "string", "title": "ID", "description": "The ID for this Rule. Used in the API to reference this query rewrite rule. Allowed characters: a-z, A-Z, 0-9, dash (-) and underscore (_)", "maxLength": 128, "pattern": "^[A-Za-z0-9_\\-]+$" }`
source	`"source": { "type": "string", "title": "Source", "description": "The source for this query rewrite rule, such as a Spark Job.", "hints": [ "hidden" ] }`
sourceType	`"sourceType": { "type": "string", "title": "Source Type", "description": "Source type", "hints": [ "hidden" ] }`
createdOn	`"createdOn": { "type": "string", "title": "Created Date/Time", "description": "Date/Time this rule was created.", "format": "date-time" }`
updatedOn	`"updatedOn": { "type": "string", "title": "Updated Date/Time", "description": "Date/Time this rule was last updated.", "format": "date-time" }`

Parameter	Schema
reviewer	`"reviewer": { "type": "string", "title": "Reviewer", "description": "Who reviewed this rewrite rule.", "hints": [ "hidden" ] }`
reviewedOn	`"reviewedOn": { "type": "string", "title": "Reviewed Date/Time", "description": "Date/Time this rule was last reviewed.", "hints": [ "hidden" ], "format": "date-time" }`
review	`"review": { "type": "string", "title": "Review", "enum": [ "approved", "denied" ], "default": "approved", "hints": [ "readonly" ] }`
deployed	`"deployed": { "type": "boolean", "title": "Deployed Flag", "description": "Flag to indicate if this rewrite rule has been deployed.", "hints": [ "hidden" ] }`
tags	`"tags": { "type": "array", "title": "Tags", "description": "Conditional tags to restrict the triggering of this query rewrite rule.", "items": { "type": "string" } }`
name Required	`"name": { "type": "string", "title": "Name", "description": "User-friendly name for this rule." }`

Parameter	Schema
description	`"description": { "type": "string", "title": "Description", "description": "Short description of what this rule does." }`
enabled	`"enabled": { "type": "boolean", "title": "Enabled", "description": "Set to false to disable this rule.", "default": true }`
matching	`"matching": { "type": "string", "title": "Match Query Using", "description": "Defines how the specified search terms will be compared to the user's query to determine whether this rule applies. 'keywords' means the query must match the keywords of the search terms value exactly. 'phrase' means the search query must contain one of the search terms values as a subphrase. 'text' means that at least one of the search terms values must match the query, but unlike 'phrase', the query only needs to match part of the value.", "enum": ["keywords", "text", "phrase"], "default": "keywords" }`
groups	`"groups": { "type": "string", "title": "Rule Group", "description": "Group for this rule." }`
effective_range	`"effective_range": { "type": "array", "title": "Effective Range", "description": "Time range when this rule is in effect.", "items": { "type": "string" } }`

Parameter	Schema
priority	"priority": { "type": "integer", "title": "Precedence", "description": "Used when multiple rules could take effect for a given query - higher number means higher precedence. For example, a rule with precedence of 10 will take effect before a rule with precedence of 3. Ties on precedence are broken based on creation time - the more recently created rule will take effect before the older rule." }
search_terms	"search_terms": { "type": "array", "title": "Search Terms", "description": "Search terms that trigger this rule.", "items": { "type": "string" } }
filters	"filters": { "type": "array", "title": "Filters", "description": "Filters used to determine if this rule triggers", "items": { "type": "object", "required": ["key", "value"], "properties": { "key": { "type": "string", "title": "Field Name" }, "value": { "type": "string", "title": "Field Value" } } } }
response_values	"response_values": { "type": "array", "title": "Set Response Values", "description": "Key/Value pairs to set in the fusion section of the response.", "items": { "type": "object", "required": ["key", "value"], "properties": { "key": { "type": "string", "title": "Field Name" }, "value": { "type": "string", "title": "Field Value" } } } }
query_profile_id_ss	"query_profile_id_ss": { "type": "array", "title": "Query Profile ID", "description": "Query Profile that triggers this rule", "items": { "type": "string" } }

Parameter	Schema
type Required	"type": { "type": "string", "title": "Type", "enum": ["response_value"], "default": "response_value", "hints": ["readonly"] }
doc_type	"doc_type": { "type": "string", "title": "Doc Type", "enum": ["rule"], "default": "rule", "hints": ["readonly"] }
display_type	"display_type": { "type": "string", "title": "Display Type", "enum": ["Response Value"], "default": "Response Value", "hints": ["readonly"] } }
additionalProperties	
additionalProperties	"additionalProperties": true

Set Facets Rule API Configurations

This rule type is used in the *Query Rewrite API*.

For a complete list of rule and rewrite types in the Query Rewrite API, see *Query Rewrite API Configurations*.

Set Facets

The Set Facets action customizes the facets returned in the query response. These facets are different from the facets set in the Query Workbench.

For example, an e-commerce website may display different facets on the homepage, the primary search page, and category pages.

General information

- Endpoint: `https://FUSION_HOST:FUSION_PORT/api/apps/APP_NAME/query-rewrite/schema/RULE_TYPE`
- Required parameters are:
  - name
  - type
- The values for the `hints` parameter include:
  - `hidden`: Do not display in the Fusion UI, but can be read and written in the schema.
  - `readonly`: Cannot be set by the user. If attempted, an error is generated and the operation fails.

Parameter	Schema
type	
type	string
properties	
id	`"id": { "type": "string", "title": "ID", "description": "The ID for this Rule. Used in the API to reference this query rewrite rule. Allowed characters: a-z, A-Z, 0-9, dash (-) and underscore (_)", "maxLength": 128, "pattern": "^[A-Za-z0-9_\\-]+$" }`
source	`"source": { "type": "string", "title": "Source", "description": "The source for this query rewrite rule, such as a Spark Job.", "hints": [ "hidden" ] }`
sourceType	`"sourceType": { "type": "string", "title": "Source Type", "description": "Source type", "hints": [ "hidden" ] }`
createdOn	`"createdOn": { "type": "string", "title": "Created Date/Time", "description": "Date/Time this rule was created.", "format": "date-time" }`
updatedOn	`"updatedOn": { "type": "string", "title": "Updated Date/Time", "description": "Date/Time this rule was last updated.", "format": "date-time" }`

Parameter	Schema
reviewer	`"reviewer": { "type": "string", "title": "Reviewer", "description": "Who reviewed this rewrite rule.", "hints": [ "hidden" ] },`
reviewedOn	`"reviewedOn": { "type": "string", "title": "Reviewed Date/Time", "description": "Date/Time this rule was last reviewed.", "hints": [ "hidden" ], "format": "date-time" }`
review	`"review": { "type": "string", "title": "Review", "enum": [ "approved", "denied" ], "default": "approved", "hints": [ "readonly" ] }`
deployed	`"deployed": { "type": "boolean", "title": "Deployed Flag", "description": "Flag to indicate if this rewrite rule has been deployed.", "hints": [ "hidden" ] }`
tags	`"tags": { "type": "array", "title": "Tags", "description": "Conditional tags to restrict the triggering of this query rewrite rule.", "items": { "type": "string" } }`

Parameter	Schema
name Required	"name": { "type": "string", "title": "Name", "description": "User-friendly name for this rule." }
description	"description": { "type": "string", "title": "Description", "description": "Short description of what this rule does." }
enabled	"enabled": { "type": "boolean", "title": "Enabled", "description": "Set to false to disable this rule.", "default": true }
matching	"matching": { "type": "string", "title": "Match Query Using", "description": "Defines how the specified search terms will be compared to the user's query to determine whether this rule applies. 'keywords' means the query must match the keywords of the search terms value exactly. 'phrase' means the search query must contain one of the search terms values as a subphrase. 'text' means that at least one of the search terms values must match the query, but unlike 'phrase', the query only needs to match part of the value.", "enum": ["keywords", "text", "phrase"], "default": "keywords" }
groups	"groups": { "type": "string", "title": "Rule Group", "description": "Group for this rule." }

Parameter	Schema
effective_range	`"effective_range": { "type": "array", "title": "Effective Range", "description": "Time range when this rule is in effect.", "items": { "type": "string" } }`
priority	`"priority": { "type": "integer", "title": "Precedence", "description": "Used when multiple rules could take effect for a given query - higher number means higher precedence. For example, a rule with precedence of 10 will take effect before a rule with precedence of 3. Ties on precedence are broken based on creation time - the more recently created rule will take effect before the older rule." }`
search_terms	`"search_terms": { "type": "array", "title": "Search Terms", "description": "Search terms that trigger this rule.", "items": { "type": "string" } }`
filters	`"filters": { "type": "array", "title": "Filters", "description": "Filters used to determine if this rule triggers", "items": { "type": "object", "required": ["key", "value"], "properties": { "key": { "type": "string", "title": "Field Name" }, "value": { "type": "string", "title": "Field Value" } } } }`
response_values	`"response_values": { "type": "array", "title": "Set Response Values", "description": "Key/Value pairs to set in the fusion section of the response.", "items": { "type": "object", "required": ["key", "value"], "properties": { "key": { "type": "string", "title": "Field Name" }, "value": { "type": "string", "title": "Field Value" } } } }`

Parameter	Schema
query_profile_id_ss	"query_profile_id_ss": { "type": "array", "title": "Query Profile ID", "description": "Query Profile that triggers this rule", "items": { "type": "string" } }
facet_data	"facet_data": { "type": "array", "title": "Facet Fields", "description": "Field specific facet parameters.", "minItems": 1, "items": { "type": "object", "required": ["fieldName"], "properties": { "fieldName": { "type": "string", "title": "Facet Field", "description": "Identifies a field to be treated as a facet." }, "selectType": { "type": "string", "title": "Select Type", "description": "Whether this facet supports single or multiple selections.", "enum": ["single", "multi"], "default": "multi" }, "type": { "type": "string", "title": "Facet Type", "description": "Either field or range facet.", "enum": ["field", "range"], "default": "field" }, "sort": { "type": "string", "title": "Facet Sort", "description": "Determines the ordering of the facet field constraints.", "enum": ["count", "index"], "default": "count" }, "rangeStart": { "type": "string", "title": "Range Start", "description": "Specifies the start of the facet range; required if type is range" }, "rangeEnd": { "type": "string", "title": "Range End", "description": "Specifies the end of the facet range; required if type is range" }, "rangeGap": { "type": "string", "title": "Range Gap", "description": "Specifies the span of the range as a value to be added to the lower bound; required if type is range" }, "minCount": { "type": "integer", "title": "Min Count", "description": "Specifies the minimum counts required for a facet field to be included in the response." }, "boostValues": { "type": "array", "title": "Boost Values", "description": "A list of values to boost, in order you want them included. Only 5 values may be boosted on legacy requests. DSL-requests have no hard limit though performance may degrade if very large numbers of facet-boosts are used.", "items": { "type": "string" } }, "buryValues": { "type": "array", "title": "Bury Values", "description": "A list of values to bury, in order you want them included.", "items": { "type": "string" } }, "suppressValues": { "type": "array", "title": "Suppress Values", "description": "A list of values to remove from the results.", "items": { "type": "string" } } } } }
type Required	"type": { "type": "string", "title": "Type", "enum": ["set_facets"], "default": "set_facets", "hints": ["readonly"] }

Parameter	Schema
doc_type	`"doc_type": { "type": "string", "title": "Doc Type", "enum": [ "rule" ], "default": "rule", "hints": [ "readonly" ] }`
display_type	`"display_type": { "type": "string", "title": "Display Type", "enum": [ "Set Facets" ], "default": "Set Facets", "hints": [ "readonly" ] } }`
additionalProperties	
additionalProperties	`"additionalProperties": true`

Set Params Rule API Configurations

This rule type is used in the *Query Rewrite API*.

For a complete list of rule and rewrite types in the Query Rewrite API, see *Query Rewrite API Configurations*.

Set Params

The Set Params action sets a parameter name and value *(https://solr.apache.org/guide/common-query-parameters.html)* to a search query.

For example, a workplace may use the name and value to display only specific filetypes in the results.

General information

- Endpoint: `https://FUSION_HOST:FUSION_PORT/api/apps/APP_NAME/query-rewrite/schema/RULE_TYPE`
- Required parameters are:
  - name
  - type
- The values for the `hints` parameter include:
  - `hidden`: Do not display in the Fusion UI, but can be read and written in the schema.
  - `readonly`: Cannot be set by the user. If attempted, an error is generated and the operation fails.

Parameter	Schema
type	
type	string
properties	
id	`"id": { "type": "string", "title": "ID", "description": "The ID for this Rule. Used in the API to reference this query rewrite rule. Allowed characters: a-z, A-Z, 0-9, dash (-) and underscore (_)", "maxLength": 128, "pattern": "^[A-Za-z0-9_\\-]+$" }`
source	`"source": { "type": "string", "title": "Source", "description": "The source for this query rewrite rule, such as a Spark Job.", "hints": [ "hidden" ] }`
sourceType	`"sourceType": { "type": "string", "title": "Source Type", "description": "Source type", "hints": [ "hidden" ] }`
createdOn	`"createdOn": { "type": "string", "title": "Created Date/Time", "description": "Date/Time this rule was created.", "format": "date-time" }`
updatedOn	`"updatedOn": { "type": "string", "title": "Updated Date/Time", "description": "Date/Time this rule was last updated.", "format": "date-time" }`

Parameter	Schema
reviewer	"reviewer": { "type": "string", "title": "Reviewer", "description": "Who reviewed this rewrite rule.", "hints": ["hidden"] }
reviewedOn	"reviewedOn": { "type": "string", "title": "Reviewed Date/Time", "description": "Date/Time this rule was last reviewed.", "hints": ["hidden"], "format": "date-time" }
review	"review": { "type": "string", "title": "Review", "enum": ["approved", "denied"], "default": "approved", "hints": ["readonly"] }
deployed	"deployed": { "type": "boolean", "title": "Deployed Flag", "description": "Flag to indicate if this rewrite rule has been deployed.", "hints": ["hidden"] }
tags	"tags": { "type": "array", "title": "Tags", "description": "Conditional tags to restrict the triggering of this query rewrite rule.", "items": { "type": "string" } }
name Required	"name": { "type": "string", "title": "Name", "description": "User-friendly name for this rule." }

Parameter	Schema
description	`"description": { "type": "string", "title": "Description", "description": "Short description of what this rule does." }`
enabled	`"enabled": { "type": "boolean", "title": "Enabled", "description": "Set to false to disable this rule.", "default": true }`
matching	`"matching": { "type": "string", "title": "Match Query Using", "description": "Defines how the specified search terms will be compared to the user's query to determine whether this rule applies. 'keywords' means the query must match the keywords of the search terms value exactly. 'phrase' means the search query must contain one of the search terms values as a subphrase. 'text' means that at least one of the search terms values must match the query, but unlike 'phrase', the query only needs to match part of the value.", "enum": ["keywords", "text", "phrase"], "default": "keywords" }`
groups	`"groups": { "type": "string", "title": "Rule Group", "description": "Group for this rule." }`
effective_range	`"effective_range": { "type": "array", "title": "Effective Range", "description": "Time range when this rule is in effect.", "items": { "type": "string" } }`

Parameter	Schema
priority	"priority": { "type": "integer", "title": "Precedence", "description": "Used when multiple rules could take effect for a given query - higher number means higher precedence. For example, a rule with precedence of 10 will take effect before a rule with precedence of 3. Ties on precedence are broken based on creation time - the more recently created rule will take effect before the older rule." }
search_terms	"search_terms": { "type": "array", "title": "Search Terms", "description": "Search terms that trigger this rule.", "items": { "type": "string" } }
filters	"filters": { "type": "array", "title": "Filters", "description": "Filters used to determine if this rule triggers", "items": { "type": "object", "required": ["key", "value"], "properties": { "key": { "type": "string", "title": "Field Name" }, "value": { "type": "string", "title": "Field Value" } } } }
response_values	"response_values": { "type": "array", "title": "Set Response Values", "description": "Key/Value pairs to set in the fusion section of the response.", "items": { "type": "object", "required": ["key", "value"], "properties": { "key": { "type": "string", "title": "Field Name" }, "value": { "type": "string", "title": "Field Value" } } } }
query_profile_id_ss	"query_profile_id_ss": { "type": "array", "title": "Query Profile ID", "description": "Query Profile that triggers this rule", "items": { "type": "string" } }

Parameter	Schema
params	"params": { "type": "array", "title": "Parameters", "description": "Parameters to set on the request, either appending or overriding.", "minItems": 1, "items": { "type": "object", "required": ["key", "value"], "properties": { "key": { "type": "string", "title": "Parameter Name" }, "value": { "type": "string", "title": "Parameter Value" }, "policy": { "type": "string", "title": "Update Policy", "enum": ["replace", "append", "remove", "default"], "default": "append" } } } }
type Required	"type": { "type": "string", "title": "Type", "enum": ["set_params"], "default": "set_params", "hints": ["readonly"] }
doc_type	"doc_type": { "type": "string", "title": "Doc Type", "enum": ["rule"], "default": "rule", "hints": ["readonly"] }
display_type	"display_type": { "type": "string", "title": "Display Type", "enum": ["Set Params"], "default": "Set Params", "hints": ["readonly"] } }
additionalProperties	
additionalProperties	"additionalProperties": true

Ground Truth Query Rewrite API Configurations

This query rewrite configuration is used in the *Query Rewrite API*.

For a complete list of rule and rewrite types in the Query Rewrite API, see *Query Rewrite API Configurations*.

Ground Truth

Ground truth or gold standard datasets are used in query relevance metrics and the ground truth jobs to define a specific set of documents.

For more information, see:

- *Ground Truth Queries*
- *Ground Truth Jobs*

General information

- Endpoint: `https://FUSION_HOST:FUSION_PORT/api/apps/APP_NAME/query-rewrite/schema/RULE_TYPE`
- Required parameter is: type
- The values for the `hints` parameter include:
  - `hidden`: Do not display in the Fusion UI, but can be read and written in the schema.
  - `readonly`: Cannot be set by the user. If attempted, an error is generated and the operation fails.

Parameter	Schema
type	
type	string
properties	
id	`"id": { "type": "string", "title": "ID", "description": "The ID for this Rule. Used in the API to reference this query rewrite rule. Allowed characters: a-z, A-Z, 0-9, dash (-) and underscore (_)", "maxLength": 128, "pattern": "^[A-Za-z0-9_\\-]+$" }`
source	`"source": { "type": "string", "title": "Source", "description": "The source for this query rewrite rule, such as a Spark Job.", "hints": [ "hidden" ] }`
sourceType	`"sourceType": { "type": "string", "title": "Source Type", "description": "Source type", "hints": [ "hidden" ] }`
createdOn	`"createdOn": { "type": "string", "title": "Created Date/Time", "description": "Date/Time this rule was created.", "format": "date-time" }`
updatedOn	`"updatedOn": { "type": "string", "title": "Updated Date/Time", "description": "Date/Time this rule was last updated.", "format": "date-time" }`

Parameter	Schema
reviewer	`"reviewer": { "type": "string", "title": "Reviewer", "description": "Who reviewed this rewrite rule.", "hints": [ "hidden" ] }`
reviewedOn	`"reviewedOn": { "type": "string", "title": "Reviewed Date/Time", "description": "Date/Time this rule was last reviewed.", "hints": [ "hidden" ], "format": "date-time" }`
review	`"review": { "type": "string", "title": "Review", "enum": [ "approved", "denied" ], "default": "approved", "hints": [ "readonly" ] }`
deployed	`"deployed": { "type": "boolean", "title": "Deployed Flag", "description": "Flag to indicate if this rewrite rule has been deployed.", "hints": [ "hidden" ] }`
tags	`"tags": { "type": "array", "title": "Tags", "description": "Conditional tags to restrict the triggering of this query rewrite rule.", "items": { "type": "string" } }`
type Required	`"type": { "type": "string", "title": "Type", "enum": [ "ground_truth" ], "default": "ground_truth", "hints": [ "readonly" ] }`

Parameter	Schema
doc_type	`"doc_type": { "type": "string", "title": "Doc Type", "enum": [ "query_rewrite" ], "default": "query_rewrite", "hints": [ "readonly" ] } }`
additionalProperties	
additionalProperties	`"additionalProperties": true`

Phrase Query Rewrite API Configurations

This query rewrite configuration is used in the *Query Rewrite API*.

For a complete list of rule and rewrite types in the Query Rewrite API, see *Query Rewrite API Configurations*.

Phrase

Phrase detection identifies phrases in your signals so that results with matching phrases can be boosted. This helps compensate for queries where phrases are not distinguished with quotation marks. For example, the query `ipad case` is rewritten as `“ipad case”~10^2`, meaning if `ipad` and `case` appear **within 10** terms (whitespace-delimited tokens) of each other, then boost the result by a factor of **2**.

Phrases are applied in the *Text Tagger stage* of the query pipeline.

General information

- Endpoint: `https://FUSION_HOST:FUSION_PORT/api/apps/APP_NAME/query-rewrite/schema/RULE_TYPE`
- Required parameter is: type
- The values for the `hints` parameter include:
  - `hidden`: Do not display in the Fusion UI, but can be read and written in the schema.
  - `readonly`: Cannot be set by the user. If attempted, an error is generated and the operation fails.

Parameter	Schema
type	
type	string
properties	
confidence	`"confidence": { "type": "number", "title": "Confidence", "description": "Confidence score from the phrase job.", "default": 1.0 }`
id	`"id": { "type": "string", "title": "ID", "description": "The ID for this Rule. Used in the API to reference this query rewrite rule. Allowed characters: a-z, A-Z, 0-9, dash (-) and underscore (_)", "maxLength": 128, "pattern": "^[A-Za-z0-9_\\-]+$" }`
source	`"source": { "type": "string", "title": "Source", "description": "The source for this query rewrite rule, such as a Spark Job.", "hints": [ "hidden" ] }`
sourceType	`"sourceType": { "type": "string", "title": "Source Type", "description": "Source type", "hints": [ "hidden" ] }`
createdOn	`"createdOn": { "type": "string", "title": "Created Date/Time", "description": "Date/Time this rule was created.", "format": "date-time" }`

Parameter	Schema
updatedOn	`"updatedOn": { "type": "string", "title": "Updated Date/Time", "description": "Date/Time this rule was last updated.", "format": "date-time" }`
reviewer	`"reviewer": { "type": "string", "title": "Reviewer", "description": "Who reviewed this rewrite rule.", "hints": [ "hidden" ] }`
reviewedOn	`"reviewedOn": { "type": "string", "title": "Reviewed Date/Time", "description": "Date/Time this rule was last reviewed.", "hints": [ "hidden" ], "format": "date-time" }`
review	`"review": { "type": "string", "title": "Review", "enum": [ "approved", "denied" ], "default": "approved", "hints": [ "readonly" ] }`
deployed	`"deployed": { "type": "boolean", "title": "Deployed Flag", "description": "Flag to indicate if this rewrite rule has been deployed.", "hints": [ "hidden" ] }`
tags	`"tags": { "type": "array", "title": "Tags", "description": "Conditional tags to restrict the triggering of this query rewrite rule.", "items": { "type": "string" } }`

Parameter	Schema
surface_form	`"surface_form": { "type": "string", "title": "Surface Form", "description": "The phrase itself" }`
phrase_count	`"phrase_count": { "type": "integer", "title": "Phrase Count", "description": "Denotes how many times this phrase was found in the source." }`
word_count	`"word_count": { "type": "integer", "title": "Word Count", "description": "Indicates how many words are included in the phrase." }`
boost	`"boost": { "type": "number", "title": "Boost Factor", "description": "The factor to use to boost this phrase in matching queries." }`
slop	`"slop": { "type": "integer", "title": "Slop Factor", "description": "Phrase slop" }`
type Required	`"type": { "type": "string", "title": "Type", "enum": [ "phrase" ], "default": "phrase", "hints": [ "readonly" ] }`

Parameter	Schema
doc_type	`"doc_type": { "type": "string", "title": "Doc Type", "enum": [ "query_rewrite" ], "default": "query_rewrite", "hints": [ "readonly" ] } }`
additionalProperties	
additionalProperties	`"additionalProperties": true`

Remove Words Query Rewrite API Configurations

This query rewrite configuration is used in the *Query Rewrite API*.

For a complete list of rule and rewrite types in the Query Rewrite API, see *Query Rewrite API Configurations*.

Remove Words

Use a Remove Words query rewrite to remove particular phrases from queries. Unlike other rewrites, Remove Words rules are entered manually and aren't generated by a job.

General information

- Endpoint: `https://FUSION_HOST:FUSION_PORT/api/apps/APP_NAME/query-rewrite/schema/RULE_TYPE`
- Required parameters are:
  - surface_form
  - trigger_phrase_ss
  - type
- The values for the `hints` parameter include:
  - `hidden`: Do not display in the Fusion UI, but can be read and written in the schema.
  - `readonly`: Cannot be set by the user. If attempted, an error is generated and the operation fails.

Parameter	Schems
type	
type	string
properties	
id	`"id": { "type": "string", "title": "ID", "description": "The ID for this Rule. Used in the API to reference this query rewrite rule. Allowed characters: a-z, A-Z, 0-9, dash (-) and underscore (_)", "maxLength": 128, "pattern": "^[A-Za-z0-9_\\-]+$" }`
source	`"source": { "type": "string", "title": "Source", "description": "The source for this query rewrite rule, such as a Spark Job.", "hints": [ "hidden" ] }`
sourceType	`"sourceType": { "type": "string", "title": "Source Type", "description": "Source type", "hints": [ "hidden" ] }`
createdOn	`"createdOn": { "type": "string", "title": "Created Date/Time", "description": "Date/Time this rule was created.", "format": "date-time" }`
updatedOn	`"updatedOn": { "type": "string", "title": "Updated Date/Time", "description": "Date/Time this rule was last updated.", "format": "date-time" }`

Parameter	Schems
reviewer	`"reviewer": { "type": "string", "title": "Reviewer", "description": "Who reviewed this rewrite rule.", "hints": [ "hidden" ] }`
reviewedOn	`"reviewedOn": { "type": "string", "title": "Reviewed Date/Time", "description": "Date/Time this rule was last reviewed.", "hints": [ "hidden" ], "format": "date-time" }`
review	`"review": { "type": "string", "title": "Review", "enum": [ "approved", "denied" ], "default": "approved", "hints": [ "readonly" ] }`
deployed	`"deployed": { "type": "boolean", "title": "Deployed Flag", "description": "Flag to indicate if this rewrite rule has been deployed.", "hints": [ "hidden" ] }`
tags	`"tags": { "type": "array", "title": "Tags", "description": "Conditional tags to restrict the triggering of this query rewrite rule.", "items": { "type": "string" } }`
surface_form Required	`"surface_form": { "type": "string", "title": "Phrase to Remove", "description": "Phrase to remove from the query when it contains any of the trigger phrases." }`

Parameter	Schems
confidence	`"confidence": { "type": "number", "title": "Confidence" }`
removeid_s	`"removeid_s": { "type": "string", "title": "Remove ID", "description": "If this is a trigger doc, holds the ID of the 'base' rewrite doc this is for.", "hints": [ "hidden", "readonly" ] }`
triggerdoc_b	`"triggerdoc_b": { "type": "boolean", "title": "Is Trigger Doc", "description": "True if this is a trigger doc tied to a particular 'base' rewrite doc (identified by the removeId field).", "hints": [ "hidden", "readonly" ] }`
trigger_phrase_ss Required	`"trigger_phrase_ss": { "type": "array", "title": "Trigger Phrases", "description": "The phrase to remove will be removed from the query when the query also contains any of these phrases.", "items": { "type": "string" } }`
type Required	`"type": { "type": "string", "title": "Type", "enum": [ "removeWords" ], "default": "removeWords", "hints": [ "readonly" ] }`
doc_type	`"doc_type": { "type": "string", "title": "Doc Type", "enum": [ "query_rewrite" ], "default": "query_rewrite", "hints": [ "readonly" ] } }`

Parameter	Schems
additionalProperties	
additionalProperties	`"additionalProperties": true`

Similar Queries (simq) Query Rewrite API Configurations

This query rewrite configuration is used in the *Query Rewrite API*.

For a complete list of rule and rewrite types in the Query Rewrite API, see *Query Rewrite API Configurations*.

simq

The similar queries table

If `query` leads to clicks on documents 1, 2, 3, and 4, and `similar_query` leads to clicks on documents 2, 3, 4, and 5, then there is sufficient overlap between the two queries to consider them similar.

A statistic is constructed to compute similarities based on overlap counts and query counts. The resulting table consists of documents whose `doc_type` value is "query_rewrite" and `type` value is "simq".

For more information, see *The similar queries table*.

General information

- Endpoint: `https://FUSION_HOST:FUSION_PORT/api/apps/APP_NAME/query-rewrite/schema/RULE_TYPE`
- Required parameter is: type
- The values for the `hints` parameter include:
  - `hidden`: Do not display in the Fusion UI, but can be read and written in the schema.
  - `readonly`: Cannot be set by the user. If attempted, an error is generated and the operation fails.

Parameter	Schema
type	
type	string
properties	
id	`"id": { "type": "string", "title": "ID", "description": "The ID for this Rule. Used in the API to reference this query rewrite rule. Allowed characters: a-z, A-Z, 0-9, dash (-) and underscore (_)", "maxLength": 128, "pattern": "^[A-Za-z0-9_\\-]+$" }`
source	`"source": { "type": "string", "title": "Source", "description": "The source for this query rewrite rule, such as a Spark Job.", "hints": [ "hidden" ] }`
sourceType	`"sourceType": { "type": "string", "title": "Source Type", "description": "Source type", "hints": [ "hidden" ] }`
createdOn	`"createdOn": { "type": "string", "title": "Created Date/Time", "description": "Date/Time this rule was created.", "format": "date-time" }`
updatedOn	`"updatedOn": { "type": "string", "title": "Updated Date/Time", "description": "Date/Time this rule was last updated.", "format": "date-time" }`

Parameter	Schema
reviewer	`"reviewer": { "type": "string", "title": "Reviewer", "description": "Who reviewed this rewrite rule.", "hints": [ "hidden" ] }`
reviewedOn	`"reviewedOn": { "type": "string", "title": "Reviewed Date/Time", "description": "Date/Time this rule was last reviewed.", "hints": [ "hidden" ], "format": "date-time" }`
review	`"review": { "type": "string", "title": "Review", "enum": [ "approved", "denied" ], "default": "approved", "hints": [ "readonly" ] }`
deployed	`"deployed": { "type": "boolean", "title": "Deployed Flag", "description": "Flag to indicate if this rewrite rule has been deployed.", "hints": [ "hidden" ] }`
tags	`"tags": { "type": "array", "title": "Tags", "description": "Conditional tags to restrict the triggering of this query rewrite rule.", "items": { "type": "string" } }`
query	`"query": { "type": "string", "title": "Query", "description": "A query that has a similar query." }`

Parameter	Schema
similar_query	`"similar_query": { "type": "string", "title": "Similar Query", "description": "Similar query" }`
query_count	`"query_count": { "type": "integer", "title": "Query Count", "description": "Query count" }`
similar_query_count	`"similar_query_count": { "type": "integer", "title": "Similar Query Count", "description": "Similar Query count" }`
similarity	`"similarity": { "type": "number", "title": "Similarity", "description": "Measure of how similar the two queries are." }`
type Required	`"type": { "type": "string", "title": "Type", "enum": [ "simq" ], "default": "simq", "hints": [ "readonly" ] }`
doc_type	`"doc_type": { "type": "string", "title": "Doc Type", "enum": [ "query_rewrite" ], "default": "query_rewrite", "hints": [ "readonly" ] } }`
additionalProperties	

Parameter	Schema
additionalProperties	`"additionalProperties": true`

Spell Query Rewrite API Configurations

This query rewrite configuration is used in the *Query Rewrite API*.

For a complete list of rule and rewrite types in the Query Rewrite API, see *Query Rewrite API Configurations*.

Spell

The Misspelling Detection feature maps misspellings to their corrected spellings. When Fusion receives a query containing a known misspelling, it rewrites the query using the corrected spelling in order to return relevant results instead of an empty or irrelevant results set.

Spelling corrections are applied in the *Text Tagger stage* of the query pipeline.

General information

- Endpoint: `https://FUSION_HOST:FUSION_PORT/api/apps/APP_NAME/query-rewrite/schema/RULE_TYPE`
- Required parameter is: type
- The values for the `hints` parameter include:
  - `hidden`: Do not display in the Fusion UI, but can be read and written in the schema.
  - `readonly`: Cannot be set by the user. If attempted, an error is generated and the operation fails.

Parameter	Schema
type	
type	string
properties	
confidence	`"confidence": { "type": "number", "title": "Confidence", "description": "Confidence score from the misspelling job.", "default": 1.0 }`
id	`"id": { "type": "string", "title": "ID", "description": "The ID for this Rule. Used in the API to reference this query rewrite rule. Allowed characters: a-z, A-Z, 0-9, dash (-) and underscore (_)", "maxLength": 128, "pattern": "^[A-Za-z0-9_\\-]+$" }`
source	`"source": { "type": "string", "title": "Source", "description": "The source for this query rewrite rule, such as a Spark Job.", "hints": [ "hidden" ] }`
sourceType	`"sourceType": { "type": "string", "title": "Source Type", "description": "Source type", "hints": [ "hidden" ] }`
createdOn	`"createdOn": { "type": "string", "title": "Created Date/Time", "description": "Date/Time this rule was created.", "format": "date-time" }`

Parameter	Schema
updatedOn	`"updatedOn": { "type": "string", "title": "Updated Date/Time", "description": "Date/Time this rule was last updated.", "format": "date-time" }`
reviewer	`"reviewer": { "type": "string", "title": "Reviewer", "description": "Who reviewed this rewrite rule.", "hints": [ "hidden" ] }`
reviewedOn	`"reviewedOn": { "type": "string", "title": "Reviewed Date/Time", "description": "Date/Time this rule was last reviewed.", "hints": [ "hidden" ], "format": "date-time" }`
review	`"review": { "type": "string", "title": "Review", "enum": [ "approved", "denied" ], "default": "approved", "hints": [ "readonly" ] }`
deployed	`"deployed": { "type": "boolean", "title": "Deployed Flag", "description": "Flag to indicate if this rewrite rule has been deployed.", "hints": [ "hidden" ] }`
tags	`"tags": { "type": "array", "title": "Tags", "description": "Conditional tags to restrict the triggering of this query rewrite rule.", "items": { "type": "string" } }`

Parameter	Schema
surface_form	`"surface_form": { "type": "string", "title": "Misspelled Term" }`
output	`"output": { "type": "array", "title": "Corrected Term", "items": { "type": "string" } }`
action	`"action": { "type": "string", "title": "Action", "description": "Action to perform", "enum": [ "expand", "replace" ] }`
type Required	`"type": { "type": "string", "title": "Type", "enum": [ "spell" ], "default": "spell", "hints": [ "readonly" ] }`
doc_type	`"doc_type": { "type": "string", "title": "Doc Type", "enum": [ "query_rewrite" ], "default": "query_rewrite", "hints": [ "readonly" ] } }`
additionalProperties	
additionalProperties	`"additionalProperties": true`

Synonym Query Rewrite API Configurations

This query rewrite configuration is used in the *Query Rewrite API*.

For a complete list of rule and rewrite types in the Query Rewrite API, see *Query Rewrite API Configurations*.

Synonym

The Synonym Detection feature generates pairs of synonyms and pairs of similar queries. Two words are considered potential synonyms when they are used in a similar context in similar queries. A query that contains a matching term is expanded to include all of its synonyms, with the original term boosted by a factor of two.

For more information, see *Synonym detection*.

General information

- Endpoint: `https://FUSION_HOST:FUSION_PORT/api/apps/APP_NAME/query-rewrite/schema/RULE_TYPE`
- Required parameters are:
  - surface_form
  - mappings
  - type
- The values for the `hints` parameter include:
  - `hidden`: Do not display in the Fusion UI, but can be read and written in the schema.
  - `readonly`: Cannot be set by the user. If attempted, an error is generated and the operation fails.

Parameter	Schema
type	
type	string
properties	
confidence	`"confidence": { "type": "number", "title": "Confidence", "description": "Confidence score from the synonym job.", "default": 1.0 } ----`
id	`"id": { "type": "string", "title": "ID", "description": "The ID for this Rule. Used in the API to reference this query rewrite rule. Allowed characters: a-z, A-Z, 0-9, dash (-) and underscore (_)", "maxLength": 128, "pattern": "^[A-Za-z0-9_\\-]+$" }`
source	`"source": { "type": "string", "title": "Source", "description": "The source for this query rewrite rule, such as a Spark Job.", "hints": [ "hidden" ] }`
sourceType	`"sourceType": { "type": "string", "title": "Source Type", "description": "Source type", "hints": [ "hidden" ] }`
createdOn	`"createdOn": { "type": "string", "title": "Created Date/Time", "description": "Date/Time this rule was created.", "format": "date-time" }`

Parameter	Schema
updatedOn	`"updatedOn": { "type": "string", "title": "Updated Date/Time", "description": "Date/Time this rule was last updated.", "format": "date-time" }`
reviewer	`"reviewer": { "type": "string", "title": "Reviewer", "description": "Who reviewed this rewrite rule.", "hints": [ "hidden" ] }`
reviewedOn	`"reviewedOn": { "type": "string", "title": "Reviewed Date/Time", "description": "Date/Time this rule was last reviewed.", "hints": [ "hidden" ], "format": "date-time" }`
review	`"review": { "type": "string", "title": "Review", "enum": [ "approved", "denied" ], "default": "approved", "hints": [ "readonly" ] }`
deployed	`"deployed": { "type": "boolean", "title": "Deployed Flag", "description": "Flag to indicate if this rewrite rule has been deployed.", "hints": [ "hidden" ] }`
tags	`"tags": { "type": "array", "title": "Tags", "description": "Conditional tags to restrict the triggering of this query rewrite rule.", "items": { "type": "string" } }`

Parameter	Schema
surface_form Required	"surface_form": { "type": "string", "title": "Surface Form", "description": "The term that has synonyms." }
count	"count": { "type": "integer", "title": "Count", "description": "How many times this term occurred in the signal data when it was discovered" }
mappings Required	"mappings": { "type": "array", "title": "Synonym Mappings", "description": "Synonyms for the surface form.", "items": { "type": "object", "properties": { "synonym": { "type": "string", "title": "Synonym" }, "context": { "type": "string", "title": "Context" }, "direction": { "type": "string", "title": "Direction", "enum": ["oneway", "symmetric"] }, "similarity": { "type": "number", "title": "Similarity" }, "count": { "type": "integer", "title": "Count" }, "category": { "type": "string", "title": "Category", "enum": ["context", "synonym", "misspelling", "acronym"] }, "suggestion": { "type": "string", "title": "Suggestion", "enum": ["context", "pair"] } } } }
type Required	"type": { "type": "string", "title": "Type", "enum": ["synonym"], "default": "synonym", "hints": ["readonly"] }
doc_type	"doc_type": { "type": "string", "title": "Doc Type", "enum": ["query_rewrite"], "default": "query_rewrite", "hints": ["readonly"] } }
additionalProperties	

Parameter	Schema
additionalProperties	`"additionalProperties": true`

Tail Query Rewrite API Configurations

This query rewrite configuration is used in the *Query Rewrite API*.

For a complete list of rule and rewrite types in the Query Rewrite API, see *Query Rewrite API Configurations*.

Tail

Head/tail analysis is:

- Also known as the Underperforming Query Rewriting feature
- Uses signals data to identify underperforming queries
- Suggests improved queries that could produce better conversion rates

When an incoming query contains a matching underperforming query, the original query is replaced by an improved query.

General information

- Endpoint: `https://FUSION_HOST:FUSION_PORT/api/apps/APP_NAME/query-rewrite/schema/RULE_TYPE`
- Required parameter is: type
- The values for the `hints` parameter include:
  - `hidden`: Do not display in the Fusion UI, but can be read and written in the schema.
  - `readonly`: Cannot be set by the user. If attempted, an error is generated and the operation fails.

Parameter	Schema
type	
type	string
properties	
id	`"id": { "type": "string", "title": "ID", "description": "The ID for this Rule. Used in the API to reference this query rewrite rule. Allowed characters: a-z, A-Z, 0-9, dash (-) and underscore (_)", "maxLength": 128, "pattern": "^[A-Za-z0-9_\\-]+$" }`
source	`"source": { "type": "string", "title": "Source", "description": "The source for this query rewrite rule, such as a Spark Job.", "hints": [ "hidden" ] }`
sourceType	`"sourceType": { "type": "string", "title": "Source Type", "description": "Source type", "hints": [ "hidden" ] }`
createdOn	`"createdOn": { "type": "string", "title": "Created Date/Time", "description": "Date/Time this rule was created.", "format": "date-time" }`
updatedOn	`"updatedOn": { "type": "string", "title": "Updated Date/Time", "description": "Date/Time this rule was last updated.", "format": "date-time" }`

Parameter	Schema
reviewer	`"reviewer": { "type": "string", "title": "Reviewer", "description": "Who reviewed this rewrite rule.", "hints": [ "hidden" ] }`
reviewedOn	`"reviewedOn": { "type": "string", "title": "Reviewed Date/Time", "description": "Date/Time this rule was last reviewed.", "hints": [ "hidden" ], "format": "date-time" }`
review	`"review": { "type": "string", "title": "Review", "enum": [ "approved", "denied" ], "default": "approved", "hints": [ "readonly" ] }`
deployed	`"deployed": { "type": "boolean", "title": "Deployed Flag", "description": "Flag to indicate if this rewrite rule has been deployed.", "hints": [ "hidden" ] }`
tags	`"tags": { "type": "array", "title": "Tags", "description": "Conditional tags to restrict the triggering of this query rewrite rule.", "items": { "type": "string" } }`
surface_form	`"surface_form": { "type": "string", "title": "Tail Query" }`

Parameter	Schema
confidence	`"confidence": { "type": "number", "title": "Confidence" }`
output	`"output": { "type": "array", "title": "Improved Query", "items": { "type": "string" } }`
type Required	`"type": { "type": "string", "title": "Type", "enum": [ "tail" ], "default": "tail", "hints": [ "readonly" ] }`
doc_type	`"doc_type": { "type": "string", "title": "Doc Type", "enum": [ "query_rewrite" ], "default": "query_rewrite", "hints": [ "readonly" ] } }`
additionalProperties	
additionalProperties	`"additionalProperties": true`

Custom Rules API

The Custom Rules API allows you to create, read, update, and delete *custom rules*.

Example

The following example will create a new custom rule type in the myFusionApp app. The new custom rule type will use a pipeline with an ID of **custom-rule-pipeline** and will have two custom text fields for **Manufacturer** and **Type**.

This assumes a pipeline called custom-rule-pipeline already exists. If that is not the case, you can add `/{id}` to the URL and use `PUT` instead of `POST` to update an existing custom rule type. See *Create Custom Rule Actions with the API* for more information on creating custom rule types.

```
curl -u USERNAME:PASSWORD -XPOST -H "Content-type:application/json" \
https://FUSION_HOST:6764/api/apps/{app-name}/custom-rule-types -d '{ "id":"{custom-
rule-type}", "pipeline_id":"{query-pipeline}", "display_type":"My Custom Rule",
"schema": { "type" : "object", "properties" : { "custom_param1" : { "type" :
"string", "title" : "Manufacturer", "description" : "The manufacturer of the
product this rule action applies to." }, "custom_param2" : { "type": "string",
"title": "Type", "description": "The type of product this rule action applies to."
} } } }'
```

Experiments API

Use the Experiments API to compare different configuration variants and determine which ones are most successful. For example, configure different variants that use different query pipelines, and then analyze and compare search activity to see which variant best meets your goals.

Experiments let you evaluate multiple *variants* which can differ from each other by pipeline, collection, search handler, request parameters, or some combination of those. An experiment uses one or more metrics to measure the performance of each variant, so they can be compared quantitatively.

Experiments are also available through the *Experiments UI* in the Fusion UI.

Experiment types

The Experiments API supports A/B (or A/B/n) tests.

A/B testing

The Experiments API in Fusion lets you set up straightforward A/B (or A/B/n) tests. An A/B test is a two-sample hypothesis test with two variants. In A/B/n testing, variant A (the first variant that is defined) is compared with B, and with each of the other variants. For example, with an A/B/C/D test, the comparisons are A/B, A/C, and A/D.

The simplest setup is to create just two variants (A and B). Variant A is customarily considered the "control" configuration, and variant B is the test configuration or "treatment" to compare with the control configuration.

It is also common to perform A/A testing, that is, to create a separate variant with exactly the same configuration as the control. A/A testing is useful for detecting any systemic errors.

You can perform both A/B and A/A testing as a part of the same experiment. To do this, create three variants: one variant that is the control (A1), a second variant with the same configuration as the control (A2), and a third variant with the treatment configuration (B). The comparisons are A1/A2 and A1/B.

For example, the following experiment definition creates an experiment with 3 variants. The first two are identical, while the third uses a different query pipeline, called `with-recommendations`:

```
{ "id": "sample-experiment", "uniqueIdParameter": "userId",
"baseSignalsCollection": "bestbuy", "variants": [ { "name": "control-a1" }, {
"name": "control-a2", }, { "name": "b", "queryPipeline": "with-recommendations", }
], "enabled": true, "metrics": [ { "type": "ctr", "name": "CTR", "primary": true },
{ "type": "conversion-rate", "name": "purchase rate", "signalType": "purchase", } ]
}
```

As users interact with this experiment, we expect the results of metrics for the first two variants to be quite similar, since they are configured identically. With a small sample size, metrics could vary somewhat based on random chance but, as more users interact with the experiment, we expect the metrics to converge on identical results. The third variant, however, will likely perform differently, and its performance will tell us whether we should be using the `with-recommendations` pipeline for all search traffic.

Query API

You can run a query through a *query profile* by specifying the profile ID and appending the request with a *Solr query string*, as in `/api/query/<id>?<solrQuery>`.

For query profile CRUD operations, see the *Query Profiles API*.

Query Pipelines API

The Query Pipelines API allows you to define parameters that should be applied to all queries processed by the system. A query pipeline consists of one or more stages. Query pipeline stages can be defined with the *Query Stages API*. See the *Query Language Cheat Sheet* for help constructing queries to send through a query pipeline.

Query Pipeline Definition Properties

Property	Description
id *Required*	Required only when creating a new pipeline.
stages *Required*	A JSON map of the stages to include with the pipeline. The stages can exist already or they can be defined while defining the pipeline. The stage must include the stage definitions: • **id.** the ID of the stage to include. • **stageParams.** Any parameters for the stage. These are only required for a stage that has been pre-defined and you wish to modify the properties. • **type.** The stage type. You can define a stage directly on a pipeline, or you can use a pre-existing stage. If you use a pre-existing stage, you must use "ref", as a reference to an existing stage. • **skip.** If the stage should be skipped during pipeline processing. By default, this is 'false', and does not need to be defined unless you would like to skip a stage in the future. If the stage supports setting other properties, as defined in the section *Query Pipeline Stages*, then those can be defined while defining the stage within the pipeline.

Examples

Get the definition for the default query pipeline:

> You can omit the `default` pipeline name to list the definition for all pipelines, or include an app name in the URL (`/api/apps/APP_NAME/query-pipelines`) to list pipelines within a specific app.

REQUEST

```
curl -u USERNAME:PASSWORD https://FUSION_HOST:FUSION_PORT/api/query-
pipelines/default
```

RESPONSE

```
{ "id" : "default", "stages" : [ { "type" : "recommendation", "id" :
"recommendation", "numRecommendations" : 10, "useFq" : true, "numSignals" : 100,
"aggrType" : "click@doc_id_s-query_s-filters_s", "skip" : false }, { "type" :
"solr-query", "id" : "solr", "skip" : false } ] }
```

Create a new query pipeline, specifying a facet stage that is new and a solr-query stage that was previously defined:

REQUEST

```
curl -u USERNAME:PASSWORD -X POST -H 'Content-Type: application/json' -d '{ "id":
"my-pipeline", "stages": [ { "type": "facet", "id": "myFacets", "fieldFacets": [ {
"field": "data_source_s" }, { "field": "author_s", "minCount": 5, "missing": true
}, { "field": "isRetweet_b" } ], "skip": false }, { "id": "mySolr", "httpMethod":
"POST", "allowFederatedSearch": false, "preferredReplicaType": "pull", "type":
"solr-query", "skip": false, "responseSignalsEnabled": true } } }'
https://FUSION_HOST:FUSION_PORT/api/apps/APP_NAME/my-pipeline/query-pipelines
```

RESPONSE

```
{ "id" : "my-pipeline", "stages" : [ { "type" : "facet", "id" : null, "fieldFacets"
: [ { "field" : "data_source_s", "prefix" : null, "sort" : null, "limit" : null,
"offset" : null, "missing" : null, "method" : null, "threads" : null, "minCount" :
null, "enumCacheMinDf" : null }, { "field" : "author_s", "prefix" : null, "sort" :
null, "limit" : null, "offset" : null, "missing" : true, "method" : null, "threads"
: null, "minCount" : 5, "enumCacheMinDf" : null }, { "field" : "isRetweet_b",
"prefix" : null, "sort" : null, "limit" : null, "offset" : null, "missing" : null,
"method" : null, "threads" : null, "minCount" : null, "enumCacheMinDf" : null } ],
"skip" : false }, { "type" : "solr-query", "id" : null, "skip" : false } ] }
```

Query a pipeline named 'default' and a collection named "docs" for the term "solr". Also limit the response to only document 'titles' and return it in JSON format:

REQUEST

```
curl -u USERNAME:PASSWORD https://FUSION_HOST:FUSION_PORT/api/query-
pipelines/default/collections/docs/select?q=solr&fl=title&wt=json
```

RESPONSE

```
{ "response": { "numFound": 106, "start": 0, "docs": [ { "title": [ "Solr and
SolrAdmin APIs - Fusion Documentation - Lucidworks" ] }, { "title": [ "Solr and
SolrAdmin APIs - Fusion Documentation - Lucidworks" ] }, { "title": [ "Search
Clusters - Fusion Documentation - Lucidworks" ] }, { "title": [ "Search Clusters -
Fusion Documentation - Lucidworks" ] }, { "title": [ "Searching - Fusion
Documentation - Lucidworks" ] } ] }, "responseHeader": { "status": 0, "QTime": 1,
"params": { "fl": "title", "lw.pipelineId": "default", "q": "solr", "wt": "json",
"rows": "5", "defType": "edismax" } } }
```

Query Profiles API

The Query Profiles API lets you manage query profiles. Query profiles allow users to change query pipelines during search time while pointing the search toward a static endpoint. This gives flexibility to test different stage combinations without reconfiguration. For more information, see *Query Profiles*.

To run a query through a query profile, use the *Query API*.

Query Stage Plugins API

The Query Stage Plugins API manages custom query stages developed using the *Query Stage SDK*.

Example: Upload a custom plugin stage

```
curl -u USERNAME:PASSWORD -X PUT -H "Content-Type:application/zip" --data-binary
@sample-plugin-stage-0.0.1.zip https://FUSION_HOST:FUSION_PORT/api/query-stage-
plugins
```

Query Stages API

The Query Stages API provides endpoints to list query stage configuration properties.

Get the properties for the "apply-defaults" type:

REQUEST

```
curl -u USERNAME:PASSWORD https://FUSION_HOST:FUSION_PORT/api/query-
stages/schema/set-params
```

RESPONSE

```
{ "type" : "object", "title" : "Additional Query Parameters", "description" : "This
stage allows you to set, append, and remove additional query parameters",
"properties" : { "skip" : { "type" : "boolean", "title" : "Skip This Stage",
"description" : "Set to true to skip this stage.", "default" : false, "hints" : [
"advanced" ] }, "label" : { "type" : "string", "title" : "Label", "description" :
"A unique label for this stage.", "hints" : [ "advanced" ], "maxLength" : 255 },
"condition" : { "type" : "string", "title" : "Condition", "description" : "Define a
conditional script that must result in true or false. This can be used to determine
if the stage should process or not.", "hints" : [ "code", "javascript", "advanced"
] }, "params" : { "type" : "array", "title" : "Parameters and Values", "items" : {
"type" : "object", "required" : [ "key" ], "properties" : { "key" : { "type" :
"string", "title" : "Parameter Name" }, "value" : { "type" : "string", "title" :
"Parameter Value" }, "policy" : { "type" : "string", "title" : "Update Policy",
"enum" : [ "replace", "append", "remove", "default" ], "default" : "append" } } } }
}, "category" : "Advanced", "categoryPriority" : 2
```

Connectors Service APIs

The Connect API are a group of APIs that can help you with configurations related to your choice of connectors. You can use them to *configure connector datasources*, *create logs of installed connector configurations*, and more.

Connector Datasources API

Use the Connector Datasources API to create and configure datasources and examine or clear items and tables from the crawl database *(https://support.lucidworks.com/hc/en-us/articles/360049671493-What-is-Fusion-CrawlDB-and-what-can-it-be-used-for-)*.

See *Connectors* and *Indexing your Data* for related information.

Working with the crawl database

Some of the connectors use a crawl database to track documents that have been seen by prior crawls and are able to use this information to understand which documents are new or have been updated or removed and take appropriate action in the index. The `/api/connectors/datasources/DATASOURCE_ID` endpoints allow looking into the crawl database and dropping tables or clearing the database.

The connectors that support the crawl database are currently `lucid.fs` and `lucid.solrxml`. The `lucid.anda` connector also uses a crawl database, but it is not the same database, and does not have a REST API or other interface to access it.

Examining a crawlDB

The output from a GET request to `/api/connectors/datasources/DATASOURCE_ID` will include several sections detailing the database structure:

- `counters`: The `counters` section reports the document counts of database activities, such as table inserts.
- `ops`: The `ops` section reports on database operations that have occurred, such as initiating tables, retrieving items, processing items and table drops.
- `tables`: The `tables` section lists the tables in the database with a count of the number of items in each table. Inspecting the items is described in the next section.

Drop tables from a crawlDB

The output from a DELETE request to `/api/connectors/datasources/DATASOURCE_ID/db/<table>` will be empty. When dropping the database, note that no documents will be removed from the index. However, the crawl database will be empty, so on the next datasource run, all documents will be treated as though they were never seen by the connectors.

When dropping tables, be aware that the `items` table does not delete documents from the index, but instead changes the database so database considers them new documents. When dropping other tables, such as the `errors` table, it will merely clear out old error messages.

Clear or delete items from a crawlDB

A CLEAR request to `/api/connectors/datasources/DATASOURCE_ID/db/items/<item>` removes the information from the *Solr Index only*. This option is useful when Solr data gets out of sync with the Crawl Database.

A DELETE request removes the information *from the Crawl Database only*. Note that this does not affect the Solr Index.

Examples

Get datasources assigned to the "demo" collection:
REQUEST

```
curl -u USERNAME:PASSWORD
https://FUSION_HOST:FUSION_PORT/api/connectors/datasources?collection=demo
```

RESPONSE

```
[ { "id" : "database", "created" : "2014-05-04T19:47:22.867Z", "modified" : "2014-05-04T19:47:22.867Z", "connector" : "lucid.jdbc", "type" : "jdbc", "description" : null, "pipeline" : "conn_solr", "properties" : { "db" : null, "commit_on_finish" : true, "verify_access" : true, "sql_select_statement" : "select CONTENTID as id from CONTENT;", "debug" : false, "collection" : "demo", "password" : "password", "url" : "jdbc:postgresql://FUSION_HOST:5432/db", "nested_queries" : null, "clean_in_full_import_mode" : true, "username" : "user", "delta_sql_query" : null, "commit_within" : 900000, "primary_key" : null, "driver" : "org.postgresql.Driver", "max_docs" : -1 } } ]
```

Create a datasource to index Solr-formatted XML files:
REQUEST

```
curl -u USERNAME:PASSWORD -X POST -H 'Content-type: application/json' -d '{ "id":"SolrXML", "connector":"lucid.solrxml", "type":"solrxml", "properties":{ "path":"/Applications/solr-4.10.2/example/exampledocs", "generate_unique_key":false, "collection":"MyCollection" } }' https://FUSION_HOST:FUSION_PORT/api/connectors/datasources
```

RESPONSE

```
{ "id" : "SolrXML", "created" : "2015-05-18T15:47:51.199Z", "modified" : "2015-05-18T15:47:51.199Z", "connector" : "lucid.solrxml", "type" : "solrxml", "properties" : { "commit_on_finish" : true, "verify_access" : true, "generate_unique_key" : false, "collection" : "MyCollection", "include_datasource_metadata" : true, "include_paths" : [ ".*\\.xml" ], "initial_mapping" : { "id" : "a35c9ff3-dbb6-434b-af40-597722c2986a", "skip" : false, "label" : "field-mapping", "type" : "field-mapping" }, "path" : "/Applications/apache-repos/solr-4.10.2/example/exampledocs", "exclude_paths" : [ ], "url" : "file:/Applications/apache-repos/solr-4.10.2/example/exampledocs/", "max_docs" : -1 } }
```

Change the `max_docs` value for the above datasource:
REQUEST

```
curl -u USERNAME:PASSWORD -X PUT -H 'Content-type: application/json' -d '{ "id":"SolrXML", "connector":"lucid.solrxml", "type":"solrxml", "properties":{ "path":"/Applications/solr-4.10.2/example/exampledocs", "max_docs":10 } }' https://FUSION_HOST:FUSION_PORT/api/connectors/datasources/SolrXML
```

RESPONSE

```
true
```

Delete the datasource named 'database':
REQUEST

```
curl -u USERNAME:PASSWORD -X DELETE
https://FUSION_HOST:FUSION_PORT/api/connectors/datasources/database
```

RESPONSE

If successful, no response.

Clear a datasource
You can use both of these APIs in order to fully clear the data:

REQUEST

```
curl -X POST 'http://FUSION_HOST:FUSION_PORT/api/solr/COLLECTION_NAME/update?
commit=true' -H 'Content-Type: application/json' --data-binary '{"delete":
{"query":"_lw_data_source_s:DATASOURCENAME"}}' `curl -X DELETE -u USERNAME:PASSWORD
'http://FUSION_HOST:FUSION_PORT/api/connectors/datasources/DATASOURCENAME/db' `
```

The first clears the data from the datasource but does not clear the crawlDB. So if you attempt to index the same document set again, indexing will skip the documents because they are still in the crawl DB. If you send the command to delete the crawlDB afterward, you can then reload the docs.

The `validate=true` element in the create datasource command only validates the datasource. It does *not* automatically save the datasource. An example using this element is:

`https://FUSION_HOST:FUSION_PORT/api/apps/APP_NAME/datasources?validate=true` *`(https://FUSION_HOST:FUSION_PORT/api/apps/APP_NAME/datasources?validate=true)`*

The `POST /datasources` section of the API specification allows you to set the `validate` element for testing and use.

Connector Plugins API

Use the Connector Plugins API to install connectors and update configurations.

Each connector has one or more plugin types. After querying for the list of installed connectors, you can use the connector names to get the list of plugins for each one. Querying a specific plugin returns its configuration schema.

How does this differ from the *Connector Plugins Repository API*?
This API is used to list plugins that are already installed on your Fusion cluster and list their configuration properties.

Examples

Get all available connectors:
REQUEST

```
curl -u USERNAME:PASSWORD https://FUSION_HOST:FUSION_PORT/api/connectors/plugins
```

RESPONSE

```
[ "lucid.twitter.search", "lucid.push", "lucid.hadoop.mapr", "lucid.azure.table",
"lucid.anda", "lucid.hadoop.cloudera", "lucid.azure.blob", "lucid.hortonworks",
"lucid.mongodb", "lucid.hadoop", "lucid.hadoop.apache2", "lucid.fs",
"lucid.hadoop.apache1", "lucid.jdbc", "lucid.hadoop.intel", "lucid.twitter.stream",
"lucid.solrxml" ]
```

List the property specifications for the "file" type of the "lucid.fs" connector (the output has been truncated for space):
REQUEST

```
curl -u USERNAME:PASSWORD
https://FUSION_HOST:FUSION_PORT/api/connectors/plugins/lucid.fs/types/file
```

RESPONSE

```
{ "props" : [ { "description" : "general", "name" : "---" }, { "read_only" : false,
"default_value" : null, "description" : "datasource.collection", "hints" : [
"summary" ], "allowed_values" : null, "name" : "collection", "type" : "string",
"required" : false }, { "read_only" : false, "default_value" : false, "description"
: "datasource.debug", "hints" : [ "advanced" ], "allowed_values" : null, "name" :
"debug", "type" : "boolean", "required" : false }, { "read_only" : false,
"default_value" : null, "description" : "datasource.initial_mapping", "hints" : [
"advanced" ], "allowed_values" : null, "name" : "initial_mapping", "type" : "map",
"required" : false }, { "read_only" : false, "default_value" : null, "description"
: "datasource.db", "hints" : [ "advanced" ], "allowed_values" : null, "name" :
"db", "type" : "map", "required" : false }, { "description" : "root path", "name" :
"---" }, { "read_only" : false, "default_value" : null, "description" :
"datasource.path", "hints" : [ "summary" ], "allowed_values" : null, "name" :
"path", "type" : "string", "required" : true }, { "description" : "Filesystem URL",
"name" : "---" }, { "read_only" : true, "default_value" : null, "description" :
"datasource.url", "hints" : [ "summary" ], "allowed_values" : null, "name" : "url",
"type" : "string", "required" : false }, ... ] }
```

Installing plugins

Successful

```
curl -u USERNAME:PASSWORD -X POST "{cluster}/connectors/plugins?id=lucidworks.fs" {
"pluginId" : "lucidworks.fs", "status" : { "action" : "INSTALL", "status" :
"SUCCESS", "message" : "Plugin lucidworks.fs was installed successfully." } }
```

Update
To update a connector, run the install `curl` command again. If the connector is already installed, the output will include `"action" : "UPDATE"`.

```
curl -u USERNAME:PASSWORD -X POST "{cluster}/connectors/plugins?id=lucidworks.fs" {
"pluginId" : "lucidworks.fs", "status" : { "action" : "UPDATE", "status" :
"SUCCESS", "message" : "Plugin lucidworks.fs was installed successfully." } }
```

Unsuccessful
If an installation is unsuccessful, the output will include `"status" : "NOOP"` and a reason for the failure. In the example below, an invalid connector `id` was given.

```
curl -u USERNAME:PASSWORD -X POST "{cluster}/connectors/plugins?id=lucidworks.asd"
{ "pluginId" : "lucidworks.asd", "status" : { "action" : "INSTALL", "status" :
"NOOP", "message" : "Plugin not found in repository: lucidworks.asd" } }
```

Connector Plugins Repository API

The Connector Plugins Respository API lists plugins from the built-in plugin repository.

> When you install a connector or plug-in on Fusion 5.x.x, you do not need to supply the .zip file. The installation obtains the .zip file from the repository. Fusion 4.x.x and earlier releases required you to supply the .zip file.

Examples

The examples below use the following parameter values:

Parameter	Value
User	`admin`
Password	`Password123`
Cluster	`http://FUSION_HOST:FUSION_PORT` *(http://FUSION_HOST:FUSION_PORT)*

Change these values if you are using the example commands.

Listing plugins

> A plugin's `version` and `sdkVersion` value depends on the connector's *platform version* and update history. You might encounter a plugins with very different `version` values, such as `2.1.0` and `5.2.0`. The variance in versioning is expected.

All plugins

```
curl -u USERNAME:PASSWORD {cluster}/connectors/repository [ { "id" :
"lucidworks.fs", "version" : "2.1.0", "sdkVersion" : "2.0.0", "location" :
"com/lucidworks/connector/plugins/lucidworks.connector.fs/2.1.0/lucidworks.connecto
r.fs-2.1.0.zip" }, { "id" : "lucid.twitter-search", "version" : "5.2.0",
"sdkVersion" : "5.2.0", "location" :
"com/lucidworks/connector/plugins/lucid.twitter-search/5.2.0/lucid.twitter-search-
5.2.0.zip" }, { ... } ]
```

Specific plugin

```
curl -u USERNAME:PASSWORD {cluster}/connectors/repository?name=lucidworks.fs [ {
"id" : "lucidworks.fs", "version" : "2.1.0", "sdkVersion" : "2.0.0", "location" :
"com/lucidworks/connector/plugins/lucidworks.connector.fs/2.1.0/lucidworks.connecto
r.fs-2.1.0.zip" } ]
```

Invalid plugin
If you attempt to list an invalid plugin name, the output will appear as follows:

```
curl -u USERNAME:PASSWORD {cluster}/connectors/repository?name=lucidworks.asd [ ]
```

Fusion Proxy APIs

This group of APIs lets you manage authentication and permissions in your Fusion instance. You can use these endpoints to control *session* keys, manage native *realms user accounts*, and more.

Realms API

Realms are used to authenticate users across several different user access control systems.

There are seven realm types currently supported:

- *Native*, which uses Fusion itself to manage users and passwords.
- *LDAP*, which uses an LDAP server as the source of truth for usernames and passwords.
- *Kerberos*, where Fusion can be configured to use Kerberos for authentication.
- *SAML*, which uses the SAML 2.0 protocol to provide single sign-on.
- *SSO Trusted HTTP*, which can be configured to return a list of group names, and then map the groups to Fusion roles in the security realm definition.
- *JSON Web Token (JWT)*, where you can configure Fusion to use a shared secret key to encrypt the JWT payload.
- *OpenID Connect*, which is an identity authorization layer that supplements the OAuth 2.0 protocol. See *Security Realms* for more information about each realm type.

Authenticating users with an LDAP system creates a user record in Fusion, which includes a property for the realm the user belongs to. This Fusion user record is used by administrators to grant users access permissions for the UI or REST API services.

Create, Update, Delete or List Realms

The path for this request is:

`/api/realm-configs/<id>`

where *<id>* is the ID of a realm. The ID is optional for a GET request and omitted from a POST request.

A GET request returns the configured realms. If ID is omitted, all realms will be returned.

A POST request creates a new realm. If the request is successful, a new ID will be generated.

A PUT request updates a realm.

A DELETE request removes the realm.

Input

Parameter	Description
name *Required*	The name of the realm. This name will appear on the login screen of the UI, and will appear in user records to identify the realm they belong to.
realmType *Required*	String value for realm type.
enabled *Required*	If **true**, the realm is available for users to use with system authentication.
ephemeralUsers	Prevents ephemeral users from being created in ZooKeeper during login. Enabling this property negates `config.autoCreateUsers`.
config.autoCreate Users	Enables/disables the auto-creation of Fusion user accounts after users successfully authenticate for the first time.
roleNames	Indicates which roles are dynamically applied to users in the realm.

Additonal Properties per Realm Type

Each realm type requires additional properties for configuration, which should be specified within the `config` object field.

See the following documentation for each realm type:

- **Native.** Native is a preconfigured security realm and Fusion manages all authentication and permissions information directly.
- **LDAP.** *Configure Fusion 5.x for LDAP*

- **Kerberos.** *Configure Fusion for Kerberos in Unix*, *Configure Fusion for Kerberos in Windows*
- **SAML.** *Configure Fusion for SAML*
- **SSO Trusted HTTP.** *Configure Fusion for SSO*
- **JWT.** *Configure Fusion for JWT*
- **OpenID Connect.** *Configure OpenID Connect Authentication*

Output

When creating a new realm, the output will include the properties for the realm just created, or an error to indicate a problem with the entry.

For a GET request, the output will include all defined properties of the realm.

For a DELETE or a PUT request, no output will be returned.

Examples

Get details of the default 'native' realm:

REQUEST

```
curl -u USERNAME:PASSWORD https://FUSION_HOST:FUSION_PORT/api/realm-
configs/86df9b5b-4a1c-4b0b-bc10-25aee55fef63
```

RESPONSE

```
{ "enabled": true, "id": "86df9b5b-4a1c-4b0b-bc10-25aee55fef63", "name": "native",
"realmType": "native" }
```

Create a realm to support LDAP authentication:

REQUEST

```
curl -u USERNAME:PASSWORD -X POST -H 'Content-type: application/json' -d '{
"realmType": "ldap", "name": "dev-ldap3", "enabled": true, "roleNames":
["developer","admin"], "config": { "autoCreateUsers": true, "host": "FUSION_HOST",
"ssl": true, "port": 10636, "ephemeralUsers": false, "login": { "bindDnTemplate":
"uid={},ou=users,dc=security,dc=example,dc=com" } } }'
https://FUSION_HOST:FUSION_PORT/api/realm-configs
```

RESPONSE

```
{ "realmType": "ldap", "id": "fd90a918-1e5c-4f27-952c-fd81a068ee85", "name": "dev-
ldap3", "enabled": true, "createdAt": "2021-07-28T15:15:04Z", "config": {
"autoCreateUsers": true, "host": "FUSION_HOST", "ssl": true, "port": 10636,
"ephemeralUsers": false, "login": { "bindDnTemplate": "uid=
{},ou=users,dc=security,dc=example,dc=com" } }, "roleNames": [ "developer", "admin"
] }
```

Roles API

Roles are groups of permissions that allow access to the UI and the REST APIs. See *Roles* for details.

Security Realms can be configured to use LDAP group membership to assign Roles to users. See the *LDAP configuration for Fusion 5.x* instructions for details.

Sessions API

The Sessions API provides a way for clients to manage Fusion sessions and cookies. When a client successfully authenticates, Fusion will create a unique identifier for the session. This identifier becomes the session key and is mapped to the client. The authentication response contains an HTTP cookie, which is typically stored by the client for later use. By sending this cookie back to Fusion, a client can securely access Fusion without having to re-authenticate.

The Sessions API is useful for performance and repeatability.
By default, sessions time out after 30 minutes of no activity.

Update the cookies timeout in the **values.yaml** file using the below configuration to set the cookie to expire in 1 hour.

```
api-gateway: jwtExpirationSecs: 3600
```

Create a Session

The path for this request is:

`/api/session?realmName=REALM_NAME`

where the query parameter *realmName* takes as its value the name of a realm to authenticate against.

In order to see this object within the *Fusion UI*, it **must** be associated with an app. To do this, create the object using the `/apps` endpoint.

Input

Parameter	Description
username *Required*	The username to use in authentication.
password *Required*	The password to use in authentication.

Output

The output will include a cookie ID in the HTTP response header. This can be saved to a file and re-used with subsequent REST API requests.

Examples

Create a session against an LDAP server and store it in a file named 'cookies':

REQUEST

```
curl -c cookies -i -H "content-type:application/json" -X POST -d
'{"username":"USERNAME", "password":"PASSWORD"}'
https://FUSION_HOST:FUSION_PORT/api/session?realmName=REALM_NAME
```

RESPONSE

```
HTTP/1.1 201 Created Set-Cookie: id=840a33d4-b650-49f2-87a4-
85412e99b37c;HttpOnly;Path=/api Content-Length: 0 Server: Jetty(9.1.4.v20140401)
```

In this case, we got a response because we set `curl` to include the HTTP in the output. Otherwise, we would not know for sure the session was created.

Use the cookie in another cURL request to see all collections:

```
curl -b cookies https://FUSION_HOST:FUSION_PORT/api/collections
```

View Session Details

To view session details, including the current user record, all roles directly assigned to that user, and all roles inherited from the realm by that user, use the following path for the request:

`/api/session`

Examples

REQUEST

```
curl 'https://FUSION_HOST:FUSION_PORT/api/session' -H 'Cookie: id=416925d6-6d26-
4afd-b31d-ced61714d287'
```

RESPONSE

```
{ "user": { "id": "abc123", "username": "foo" }, "roles": [ {"name": "bar"} ] }
```

End a Session

The path for this request is:

`/api/session`

Examples

Sessions can be ended automatically in the Fusion UI by clicking **Account Settings** **> Log out**.

User API

The User API allows you to create, update, and remove user accounts. This API should only be called to manage users in the native security realm. Users from other security realms are managed directly by Fusion's auth proxy.

Indexing Service APIs

The Index APIs are a group of APIs that can help you manage the way your Fusion instance processes data. You can use them to *split incoming data*, *list pipelines*, *store blobs* in the Solr database, and more.

Index Pipelines API

The Index Pipelines API provides methods for managing a set of named index pipelines. Every pipeline is made up of one or more stages. Stages can be defined during the creation of a pipeline, or stages can be defined separately and included into one or more pipelines. For details of the REST API for index stages, see *Index Stages API*.

Document processing proceeds stage by stage in a linear fashion. The order of the stages in a pipeline is the order in which they were defined. At installation, Fusion includes several pre-configured pipelines. See *Index Pipelines* for details on these default pipelines.

For more information about structuring documents for indexing, see *Importing Data with the REST API*.

Examples

> The full set of fields for each stage type are not included in the following examples. See *Index Pipeline Stages Reference* for more information about the fields available for each stage.

List the 'default' pipeline:

REQUEST

```
curl -u USERNAME:PASSWORD https://FUSION_HOST:FUSION_PORT/api/index-pipelines/default
```

RESPONSE

```
{ "id" : "default", "stages" : [ { "type" : "solr-index", "id" : "solr-default", "skip" : false } ] }
```

Create the pipeline in the "MyApp" app

REQUEST

```
curl -u USERNAME:PASSWORD -X POST -H 'Content-type: application/json' -d '{ "id": "my-index-pipeline", "stages": [ { "id": "field_mapping", "type": "field-mapping", "reservedFieldsMappingAllowed": false }, { "id": "solr_dynamic_field_mapping", "type": "solr-dynamic-field-name-mapping", "advancedTextFieldsIndexing": true }, { "id": "solr_indexing", "type": "solr-index", "fieldsList": false } ] }' https://FUSION_HOST:FUSION_PORT/api/apps/MyApp/index-pipelines
```

RESPONSE

```
{ "id": "my-index-pipeline", "stages": [ { "id": "field_mapping",
"translationMappings": [], "retentionMappings": [], "updateMappings": [],
"unmappedRule": { "keep": true, "delete": false, "fieldToMoveValuesTo": "",
"fieldToCopyValuesTo": "", "valueToAddToUnmappedFields": "",
"valueToSetOnUnmappedFields": "" }, "reservedFieldsMappingAllowed": false, "type":
"field-mapping", "skip": false, "secretSourceStageId": "field_mapping" }, { "id":
"solr_dynamic_field_mapping", "duplicateSingleValuedFields": false,
"fieldsToExclude": [], "advancedTextFieldsIndexing": true,
"maxTextLengthForAdvancedIndexing": 100, "type": "solr-dynamic-field-name-mapping",
"skip": false, "secretSourceStageId": "solr_dynamic_field_mapping" }, { "id":
"solr_indexing", "enforceSchema": true, "fieldsList": false, "dateFormats": [],
"params": [], "bufferDocsForSolr": true, "allowBlacklistedRequestParameters":
false, "type": "solr-index", "skip": false, "secretSourceStageId": "solr_indexing"
} ], "properties": { "secretSourcePipelineId": "my-index-pipeline" } }
```

Index Profiles API

These endpoints are used to manage *index profiles*. To send a document to an index profile, see the *Indexing Service APIs*.

Index Stages API

The Index Stages API provides endpoints to list index stage configuration properties.

An index pipeline is comprised of index stages. Each index stage has a name and a type. The name identifies the stage instance, and the type identifies its class. Every stage type has a number of properties, which can be configured for a particular index stage instance. See the section *Index Pipeline Stages* for a taxonomy of index stage types.

View the configuration properties for index stage type "regex-extractor":

REQUEST

```
curl -u USERNAME:PASSWORD https://FUSION_HOST:FUSION_PORT/api/index-
stages/schema/regex-extractor
```

RESPONSE

```
{ "type" : "object", "title" : "Regex Field Extraction", "description" : "This
stage allows you to extract entities using regular expressions", "properties" : {
"rules" : { "type" : "array", "title" : "Regex Rules", "items" : { "type" :
"object", "required" : [ "pattern" ], "properties" : { "source" : { "type" :
"array", "title" : "Source Fields", "items" : { "type" : "string" } }, "target" : {
"type" : "string", "title" : "Target Field" }, "pattern" : { "type" : "string",
"title" : "Regex Pattern", "format" : "regex" }, "annotateAs" : { "type" :
"string", "title" : "Annotation Name" } } } } } }
```

Parsers API

The Parser API provides CRUD operations for *parsers*.

To configure an index pipeline to use a specific parser, see the *Index Pipelines API*.

	In order to see this object within the *Fusion UI*, it **must** be associated with an app. To do this, create the object using the `/apps` endpoint.

For more information about structuring documents for indexing, see *Importing Data with the REST API*.

Job Launcher APIs

The Job Launcher API configures and launches the Spark driver pod for running Spark jobs.

Job Rest Server APIs

This group of APIs allows you to manage services within a Fusion instance and organize the data you collect and the way you store it. You can use them to *generate personalized lists* of potential interests, define *schedules for jobs*, and more.

Recommendations API

The Recommendations REST API uses signals and aggregated signals to return a list of items that can be used for recommendations.

To use the REST API Recommendations service to get recommendations for items in some collection, that collection must have associated signals and aggregated-signals system collections. How good the recommendations are depends on how well the information in the signals and aggregated signals collections, which is derived from observed user behavior, matches user behavior going forward.

In addition to these endpoints, is also possible to get recommendations as part of a query request.

See *Recommendations and Boosting* for a discussion of when to use the API and when to use recommender query stages.

Recommendation types

The API includes separate endpoints for retrieving different types of recommendations:

`itemsForQuery`	Get items for a defined query. The response is a list of *document IDs* and their weights.
`queriesForItem`	Get queries for a defined item (a document ID). This finds the top queries that led users to the defined item. The response is a list of *query terms* and their weights.

Output

The output includes the following sections:

`header`	The query parameters (in a section named `queryParams`) and the total time it took to process the query.
`items`	Depending on the recommendation type: • `itemsForQuery` The document IDs and the weights of aggregated events that match the query. This type supports a `debug` option that adds a `debug` section to the output. • `queriesForItem` The queries and the weights of aggregated events that match the document ID.

Examples

Below are examples for each recommendation type.

itemsForQuery

Get the top items for the query 'ipod':

INPUT

```
curl -u USERNAME:PASSWORD
https://FUSION_HOST:FUSION_PORT/api/recommend/lucidworks102/itemsForQuery?
q=ipod&fq=count_d:4&debug=true
```

OUTPUT

```
{ "header" : { "queryParams" : { "aggrType" : "*", "rows" : 10, "collection" :
"lucidworks102", "aggrRows" : 100, "debug" : true, "q" : "ipod", "fq" : [
"count_d:4" ] }, "totalTime" : 5 }, "items" : [ { "weight" : 1.0726584E-11, "docId"
: "8771929" }, { "weight" : 3.865899E-12, "docId" : "9225439" }, { "weight" :
9.230597E-12, "docId" : "3109302" } ], "debugInfo" : { "aggrTime" : 1, "queryTime"
: 4, "solrParams" : { "mm" : [ "50%" ], "pf" : [ "query_t^3", "query_t~2^7",
"query_t~0^1" ], "fl" : [ "id", "doc_id_s", "weight_d" ], "sort" : [ "score
desc,weight_d desc" ], "q" : [ "ipod" ], "qf" : [ "query_t" ], "collection" : [
"lucidworks102_signals_aggr" ], "fq" : [ "aggr_type_s:*", "count_d:4" ], "rows" : [
"100" ], "defType" : [ "edismax" ] } } }
```


queriesForItem

INPUT

```
curl -u USERNAME:PASSWORD
https://FUSION_HOST:FUSION_PORT/api/recommend/lucidworks102/queriesForItem?
docId=9225439
```

OUTPUT


```
{ "header" : { "queryParams" : { "aggrType" : "*", "rows" : 10, "collection" :
"lucidworks102", "docId" : "9225439" }, "totalTime" : 8 }, "items" : [ { "query" :
"ipod", "weight" : 3.865899E-12 }, { "query" : "columbusday ipod mp3 20111009",
"weight" : 3.5141304E-12 }, { "query" : "apple itouch", "weight" : 2.3619889E-12 },
{ "query" : "ipod 4th generation", "weight" : 1.6436526E-12 }, { "query" : "ipod
touch 4th generation", "weight" : 9.674966E-13 }, { "query" : "onlinemidnightsale
ipod mp3players", "weight" : 9.568035E-13 }, { "query" : "ipod touch", "weight" :
7.774231E-13 }, { "query" : "itouch", "weight" : 7.707221E-13 } ] }
```

Signals API

The Signals API accepts a set of *signals*, encoded as JSON objects, for indexing into a signals collection.

Normally, signals are indexed just like ordinary documents, through a configured datasource and index pipeline. This API is provided for cases where it is more convenient to index signals directly.

To aggregate signals, use a *SQL aggregation* job, which is a kind of *Spark job*.

	You can manage aggregation jobs in the *Jobs manager* and *Scheduler* in the Admin UI, or with the *Spark jobs API*. We recommend using the Admin UI.

See *Signal types and structure* to learn how to send well-formed signals to this API.

Examples

Send two signal events to record user clicks:

REQUEST

	This example uses the `commit=true` setting, which commits after every signal. The setting is not recommended in production and performance testing because it negatively impacts performance. Instead of sending the `commit` parameter, consider using no commit parameter and let the defaults of the Solr collection dictate the commit frequency. Alternatively, send the `commitWithin` parameter when shorter read visibility is needed, for example, `commitWithin=3000`.

```
curl \ -u USERNAME:PASSWORD -X POST -H 'Content-type:application/json' -d @- \
https://FUSION_HOST:FUSION_PORT/api/apps/APP_NAME/signals/COLLECTION_NAME?
async=false&commit=true \ <<EOF [ {"params": { "query": "Televisiones Panasonic 50
pulgadas", "filterQueries": ["cat00000","abcat0100000", "abcat0101000",
"abcat0101001"], "docId": "2125233" }, "type":"click", "timestamp": "2011-09-
01T23:44:52.53Z" }, {"params": { "query": "Sharp", "filterQueries": ["cat00000",
"abcat0100000", "abcat0101000", "abcat0101001"], "docId": "2009324" },
"type":"click", "timestamp": "2011-09-05T12:25:37.42Z" } ] EOF
```

A successful request results in events being added to the signals collection. For the above example, the events will be represented as follows:

```
{ "responseHeader":{ "status":0, "QTime":1, "params":{ "indent":"true", "q":"docId:
2125233", "wt":"json"} }, "response":{"numFound":1198,"start":0,"docs":[ { "id":
"7aee7b1f-5cde-4957-b73c-c15881f559ec", "filters_s": "abcat0100000 $ abcat0101000 $
abcat0101001 $ cat00000", "query_orig_s": "Televisiones Panasonic 50 pulgadas",
"params.user_s": "000000df17cd56a5df4a94074e133c9d4739fae3", "docId": "2125233",
"params.query_time__s": "2011-09-01T23:43:59.75Z", "query_t": "televisiones
panasonic 50 pulgadas", "query_s": "televisiones panasonic 50 pulgadas",
"filters_orig_ss": [ "abcat0100000", "abcat0101000", "abcat0101001", "cat00000" ],
"flag_s": "EVENT", "type_s": "click", "attr_params.filterQueries_": [ "cat00000",
"abcat0100000", "abcat0101000", "abcat0101001" ], "timestamp_dt": "2011-09-
01T23:44:52.53Z", "_version_": 1478892846857584600 }, { "id": "6789a209-f5b5-457e-
9df6-8033b8f7f317", "filters_s": "abcat0100000 $ abcat0101000 $ abcat0101001 $
cat00000", "query_orig_s": "Sharp", "params.user_s":
"000001928162247ffaf63185cd8b2a244c78e7c6", "docId": "2009324",
"params.query_time__s": "2011-09-05T12:25:01.18Z", "query_t": "sharp", "query_s":
"sharp", "filters_orig_ss": [ "abcat0100000", "abcat0101000", "abcat0101001",
"cat00000" ], "flag_s": "EVENT", "type_s": "click", "attr_params.filterQueries_": [
"cat00000", "abcat0100000", "abcat0101000", "abcat0101001" ], "timestamp_dt":
"2011-09-05T12:25:37.42Z", "_version_": 1478892846859681800 } ] } }
```

Spark Jobs API

The Sparks Job API is a set of endpoints for configuring and running Spark jobs.

Spark job subtypes

For the Spark *job type*, the available subtypes are listed below.

- *SQL Aggregation job*
  A Spark SQL aggregation job where user-defined parameters are injected into a built-in SQL template at runtime.
- *Custom Python job*
  The Custom Python job provides user the ability to run Python code via Fusion. This job supports Python 3.6+ code.
- *Script*
  Run a custom Scala script as a Fusion job.

See *Additional Spark jobs* for more information.

Spark Configuration Properties

Fusion passes all configuration properties with prefix "spark." to the Spark master, Spark worker and each Spark application, both for aggregation jobs and custom-scripted processing.

These properties are stored in Fusion's ZooKeeper and can be updated via requests to Fusion endpoint `api/configurations` which will update the stored value without restarting the service, therefore existing jobs and SparkContexts will not be affected. The Fusion endpoint `api/configurations` returns all configured properties for that installation. You can examine spark default configurations in a Unix shell using the utilities `curl` and `grep`. Here is an example that checks a local Fusion installation running on port FUSION_PORT:

```
curl -u USERNAME:PASSWORD https://FUSION_HOST:FUSION_PORT/api/configurations | grep
'"spark.' "spark.executor.memory" : "2g", "spark.task.maxFailures" : "10",
"spark.worker.cleanup.appDataTtl" : "7200", "spark.worker.cleanup.enabled" :
"true", "spark.worker.memory" : "2g",
```

The default SparkContext that Fusion uses for aggregation jobs can be assigned a fraction of cluster resources (executor memory and/or available CPU cores). This allows other applications (such as scripted jobs, or shell sessions) to use the remaining cluster resources even when some aggregation jobs are running. Fusion also supports dynamic allocation for all applications. This can be overridden per application. In practice, this means that even when there is an already running SparkContext with a relatively long idle time (eg. 10 minutes) but there are no active jobs that use it, its resources (CPU cores and executor memory) will be released for use by other applications.

For scripted Spark jobs, users can specify per-job configuration overrides as a set of key / value pairs in a "sparkConfig" property element of a script job configuration, which takes precedence over values stored in ZooKeeper. The following is an example of a scripted job with a "sparkConfig" section:

```
{ "id": "scripted_job_example", "script": "val rdd =
sc.textFile(\"/foo.txt\")\nrdd.count\n", "sparkConfig": { "spark.cores.max": 2,
"spark.executor.memory": "1g" } }
```

The following table lists those Spark configuration properties that Fusion overrides or uses in order to determine applications' resource allocations.

Property	Description
`spark.master.url`	By default, left unset. This property is only specified when using an external Spark cluster; when Fusion is using its own standalone Spark cluster, this property is not set.
`spark.cores.max`	The maximum number of cores across the cluster assigned to the application. If not specified, there is no limit. The default is unset, i.e., an unlimited number of cores.
`spark.executor.memory`	Amount of memory assigned to each application's executor. The default is 2G.
`spark.scheduler.mode`	Controls how tasks are assigned to available resources. Can be either 'FIFO' or 'FAIR'. Default value is 'FAIR'.
`spark.dynamicAllocation.enabled`	Boolean - whether or not to enable dynamic allocation of executors. Default value is 'TRUE'.
`spark.shuffle.service.enabled`	Boolean - whether or not to enable internal shuffle service for standalone Spark cluster. Default value is 'TRUE'.
`spark.dynamicAllocation.executorIdleTimeout`	Number of seconds after which idle executors are removed. Default value is '60s'.
`spark.dynamicAllocation.minExecutors`	Number of executors to leave running even when idle. Default value is 0.
`spark.eventLog.enabled`	Boolean - whether or not event log is enabled. Event log stores job details and can be accessed after application finishes. Default value is 'TRUE'.
`spark.eventLog.dir`	Directory that stores event logs. Default location is `$FUSION_HOME/var/spark-eventlog`.
`spark.eventLog.compress`	Boolean - whether or not to compress event log data. Default value is 'TRUE'.
`spark.logConf`	Boolean - whether or not to log effective SparkConf of new SparkContext-s. Default value is 'TRUE'.
`spark.deploy.recoveryMode`	Default value is 'ZOOKEEPER'
`spark.deploy.zookeeper.url`	ZooKeeper connect string. Default value is `$FUSION_ZK`
`spark.deploy.zookeeper.dir`	ZooKeeper path, default value is `/lucid/spark`
`spark.worker.cleanup.enabled`	Boolean - whether or not to periodically cleanup worker data. Default value is 'TRUE'.
`spark.worker.cleanup.appDataTtl`	Time-to-live in seconds. Default value is 86400 (24h).
`spark.deploy.retainedApplications`	The maximum number of applications to show in the UI. Default value is 50.

Property	Description
`spark.deploy.retainedDrivers`	The maximum number of drivers. Default value is 50.
`spark.worker.timeout`	The maximum timeout in seconds allowed before a worker is considered lost. The default value is 30.
`spark.worker.memory`	The maximum total heap allocated to all executors running on this worker. Defaults to value of the executor memory heap.

Fusion Configuration Properties

Property	Description
`fusion.spark.master.port`	Spark master job submission port. Default value is 8766.
`fusion.spark.master.ui.port`	Spark master web UI port. Default value is 8767.
`fusion.spark.idleTime`	Maximum idle time in seconds, after which the application (ie. SparkContext) is shut down. Default value is 300.
`fusion.spark.executor.memory.min`	Minimum executor memory in MB. Default value 450Mb, which is sufficient to let Fusion components in application task's to initialize themselves
`fusion.spark.executor.memory.fraction`	A float number in range `(0.0, 1.0]` indicating what portion of `spark.executor.memory` to allocate to this application. Default value is 1.0.
`fusion.spark.cores.fraction`	A float number in range `(0.0, 1.0]` indicating what portion of `spark.cores.max` to allocate to this application. Default value is 1.0.

Machine Learning Model Service APIs

Use these APIs to generate predictions from ML models.

ML Model API

The ML Model API lets you list available models and run predictions. The ML Model service is a model prediction API for MLeap/Spark-ML models, as well as pass-through to Seldon Core-based models.

Serialized Model API

API Objective: More info here.

SQL Service APIs

The Fusion Catalog is a collection of one or more analytics projects, and each project is a collection of data assets, such as tables or relations. Fusion comes with a built-in project called "fusion."

The Fusion SQL Service API provides access to assets by data analysis applications that can perform SQL or Solr queries. It includes endpoints for finding, retrieving, and manipulating projects and assets using basic keyword and metadata-driven search.

By default, non-admin Fusion users do not have access to Catalog objects. However, the Catalog API itself does not enforce any permissions, so a user who bypasses the auth proxy has full access to all projects and assets. An admin can grant permissions to Catalog endpoints for users; see *Access Control*.

	Only certain SQL clauses are allowed: LIMIT, ORDER, and PREDICATE. Any others are rejected.

Be aware there is a hard limit of 25,000 documents returned for any SQL command. If the collection the table is generating from has more than that, the query will still return the first 25,000 documents. If you want to return fewer documents, you can use SQL LIMIT.

Intra-shard splits

If your Spark cluster has more available executor slots than the number of shards, then you can increase parallelism when reading from Solr by splitting each shard into sub-ranges using a split field. The sub range splitting enables faster fetching from Solr by increasing the number of tasks in Solr. This should only be used if there are enough computing resources in the Spark cluster.

Shard splitting is enabled by default, with two sub-ranges per shard. See *Configuration options* below for shard splitting parameters.

Body attributes

For PUT and POST requests, these are valid JSON body attributes:

Name	Type	Description
`projectId`	String	The project name
`name`	String	The asset name
`assetType`	DataAssetType	One of: + * project * table * relation * field * udf * metric
`description`	String	A string describing this asset
`sourceUri`	String	A URI to the data source
`owner`	String	The user that owns the asset
`ownerEmail`	String	The owner's email address
`tags`	Set<String>	A set of arbitrary category strings
`format`	String	The format of the underlying data source
`options`	List<String>	A list of options for the underlying data source. See *Configuration options* below for valid options.
`filters`	List<String>	A set of Solr query parameters to filter the request
`sql`	String	A SQL statement to execute
`cacheOnLoad`	boolean	'True' to cache the dataset in Spark on catalog project initialization
`dependsOn`	List<String>	A list of other assets to load before initializing this data asset
`createdOn`	Date	The asset's creation date, in ISO-8601 format; otherwise the current timestamp is used

Configuration options

Name	Description	Default
`collection`	The Solr collection name.	None
`zkhost`	A ZooKeeper connect string is the list of all servers and ports for the current ZooKeeper cluster. For example, if running a single-node Fusion developer deployment with embedded ZooKeeper, the connect string is `FUSION_HOST:9983/lwfusion/3.1.0/solr`. If you have an external 3-node ZooKeeper cluster running on servers "zk1.acme.com", "zk2.acme.com", "zk3.acme.com", all listening on port 2181, then the connect string is `zk1.acme.com:2181,zk2.acme.com:2181,zk3.acme.com:2181`	The `connectString` of the default *search cluster*
`query`	A Solr query that limits the rows to load into Spark. For example, to only load documents that mention "solr": `options("query","body_t:solr")`	`*:*`

`fields`	A subset of fields to retrieve for each document in the results, such as:
	`options("fields","id,author_s,favorited_b,…")`
	You can also specify an alias for a field using Solr's field alias syntax, such as `author:author_s`. If you want to invoke a function query, such as `rord()`, then you'll need to provide an alias, such as `ord_user:ord(user_id)`. If the return type of the function query is something other than `int` or `long`, then you'll need to specify the return type after the function query, such as:
	`foo:div(sum(x,100),max(y,1)):double`
	If you request Solr function queries, then the library must use the `/select` Solr handler to make the request as exporting function queries through `/export` is not supported by Solr.
		By default, all stored fields for each document are pulled back from Solr.

`rows`	The number of rows to retrieve from Solr per request; do not confuse this with `max_rows` (see below). This is not the maximum number of rows to read from Solr. All matching rows on the backend are read. The rows parameter is the page size. Behind the scenes, the implementation uses either deep paging cursors or Streaming API and response streaming, so it is usually safe to specify a large number of rows. By default, the implementation uses 1000 rows but if your documents are smaller, you can increase this to 10000. Using too large a value can put pressure on the Solr JVM's garbage collector. Example: `options("rows","10000")`	`1000`

Name	Description	Default
`max_rows`	The maximum number of rows; only applies when using the `/select` handler. The library will issue the query from a single task and let Solr do the distributed query processing. No paging is performed, that is, the `rows` param is set to `max_rows` when querying. Consequently, this option should not be used for large `max_rows` values, rather you should just retrieve all rows using multiple Spark tasks and then re-sort with Spark if needed. Example: `options("max_rows", "100")`	None
`request_handler`	Set the Solr request handler for queries. This option can be used to export results from Solr via /export handler which streams data out of Solr. See Exporting Result Sets for more information. The /export handler needs fields to be explicitly specified. Please use the fields option or specify the fields in the query. Example: `options("request_handler", "/export")`	`/select`
`splits`	Enable shard splitting on default field *version*. Example: `options("splits", "true")` The above option is equivalent to `options("split_field", "version")`	False
`split_field`	The field to split on can be changed using split_field option. Example: `options("split_field", "id")`	*version*
`splits_per_shard`	Split the shard into evenly-sized splits using filter queries. You can also split on a string-based keyword field but it should have sufficient variance in the values to allow for creating enough splits to be useful. In other words, if your Spark cluster can handle 10 splits per shard, but there are only 3 unique values in a keyword field, then you will only get 3 splits. Keep in mind that this is only a hint to the split calculator and you may end up with a slightly different number of splits than what was requested. Example: `options("splits_per_shard", "30")`	20
`flatten_multivalued`	Flatten multi-valued fields from Solr. Example: `options("flatten_multivalued", "false")`	true
`dv`	Fetch the docValues that are indexed but not stored by using function queries. Should be used for Solr versions lower than 5.5.0. Example: `options("dv", "true")`	false
`sample_seed`	Read a random sample of documents from Solr using the specified seed. This option can be useful if you just need to explore the data before performing operations on the full result set. By default, if this option is provided, a 10% sample size is read from Solr, but you can use the `sample_pct` option to control the sample size. Example: `options("sample_seed", "5150")`	None
`sample_pct`	The size of a random sample of documents from Solr; use a value between 0 and 1. Example: `options("sample_pct", "0.05")`	0.1

Name	Description	Default
`skip_non_dv`	Skip all fields that are not docValues. Example: `options("skip_non_dv", "true")`	false

Examples

Define a "movielens" project

```
curl -u USERNAME:PASSWORD -X POST -H "Content-type:application/json"\ -d '{ "name":
"movielens", "assetType": "project", "description": "tables and views for the
movielens project", "tags": ["movies","users"], "cacheOnLoad": false }'
"https://FUSION_HOST:FUSION_PORT/api/catalog"
```

Add a "ratings" table to the "movielens" project:

```
curl -u USERNAME:PASSWORD -X POST -H "Content-type:application/json" -d '{ "name":
"ratings", "assetType": "table", "projectId": "movielens", "description": "movie
ratings data", "tags": ["movies"], "format": "solr", "cacheOnLoad": true,
"options": ["collection -> movielens_ratings", "fields ->
user_id,movie_id,rating,rating_timestamp"] }'
"https://FUSION_HOST:FUSION_PORT/api/catalog/movielens/assets"
```

Issue a SQL statement against the "ratings" table

```
curl -u USERNAME:PASSWORD -X POST -H "Content-type:application/json" -d '{ "name":
"ratings", "assetType": "table", "projectId": "movielens", "description": "movie
ratings data", "tags": ["movies"], "format": "solr", "cacheOnLoad": true,
"options": ["collection -> movielens_ratings", "fields ->
user_id,movie_id,rating,rating_timestamp"] }'
"https://FUSION_HOST:FUSION_PORT/api/catalog/movielens/query"
```

Issue a SQL query against the "movielens" project

```
curl -u USERNAME:PASSWORD -X POST -H "Content-Type:application/json" -d '{
"sql":"SELECT m.title as title, solr.aggCount as aggCount FROM movies m INNER JOIN
(SELECT movie_id, COUNT(*) as aggCount FROM ratings WHERE rating >= 4 GROUP BY
movie_id ORDER BY aggCount desc LIMIT 10) as solr ON solr.movie_id = m.movie_id
ORDER BY aggCount DESC" }'
https://FUSION_HOST:FUSION_PORT/api/catalog/movielens/query
```

Load a catalog table from a Postgres database

```
curl -u USERNAME:PASSWORD -X POST -H "Content-type:application/json" -d '{
"projectId": "nyc_taxi", "assetType": "table", "name": "trips", "sourceUri":
"http://www.nyc.gov/html/tlc/html/about/trip_record_data.shtml", "owner": "Joe
Example", "ownerEmail": "examplejoe@gmail.com", "description": "The NYC taxi trip
data stored in Postgres using tools provided by
https://github.com/toddwschneider/nyc-taxi-data", "tags": ["nyc", "taxi",
"postgres", "trips"], "format": "jdbc", "cacheOnLoad": true, "options": ["url ->
${nyc_taxi_jdbc_url}","dbtable -> trips","partitionColumn -> id","numPartitions ->
4","lowerBound -> 0", "upperBound -> $MAX(id)", "fetchSize -> 1000"], "filters":
["pickup_latitude >= -90 AND pickup_latitude <= 90 AND pickup_longitude >= -180 AND
pickup_longitude <= 180", "dropoff_latitude >= -90 AND dropoff_latitude <= 90 AND
dropoff_longitude >= -180 AND dropoff_longitude <= 180"], "sql": "SELECT
id,cab_type_id,vendor_id,pickup_datetime,dropoff_datetime,store_and_fwd_flag,rate_c
ode_id,passenger_count,trip_distance,fare_amount,extra,mta_tax,tip_amount,tolls_amo
unt,ehail_fee,improvement_surcharge,total_amount,payment_type,trip_type,
concat_ws(',',pickup_latitude,pickup_longitude) as pickup,
concat_ws(',',dropoff_latitude,dropoff_longitude) as dropoff FROM trips" }'
"https://FUSION_HOST:FUSION_PORT/api/catalog/nyc_taxi/assets"
```

Create a data asset using a streaming expression

```
curl -u USERNAME:PASSWORD -X POST -H "Content-type:application/json" -d '{ "name":
"movie_ratings", "assetType": "table", "projectId": "movielens", "description":
"movie ratings data", "tags": ["movies"], "format": "solr", "cacheOnLoad": true,
"options": ["collection -> movielens_ratings", "expr ->
hashJoin(search(movielens_ratings,q=\"*:*\",fl=\"movie_id,user_id,rating\",sort=\"m
ovie_id
asc\",qt=\"\/export\",partitionKeys=\"movie_id\"),hashed=search(movielens_movies,q=
\"*:*\",fl=\"movie_id,title\",sort=\"movie_id
asc\",qt=\"\/export\",partitionKeys=\"movie_id\"),on=\"movie_id\")"] }'
"https://FUSION_HOST:FUSION_PORT/api/catalog/movielens/assets"
```

Send a Solr query

```
curl -u USERNAME:PASSWORD -X POST -H "Content-Type:application/json" -d '{
"solr":"*:*", "requestHandler":"/select", "collection":"movielens_movies",
"params":{ "facet":"on", "facet.field":"genre", "rows":0 } }'
https://FUSION_HOST:FUSION_PORT/api/catalog/movielens/query
```

Send a Solr query using a streaming expression

```
curl -u USERNAME:PASSWORD -X POST -H "Content-Type:application/json" --data-binary
@streaming_join.json https://FUSION_HOST:FUSION_PORT/api/catalog/movielens/query {
"solr":"hashJoin(search(movielens_ratings, q=*:*, qt=\"/export\",
fl=\"user_id,movie_id,rating\", sort=\"movie_id asc\", partitionKeys=\"movie_id\"),
hashed=search(movielens_movies, q=*:*, fl=\"movie_id,title\", qt=\"/export\",
sort=\"movie_id asc\",partitionKeys=\"movie_id\"),on=\"movie_id\")",
"collection":"movielens_ratings", "requestHandler":"/stream" }
```

Templating API

Used for template operations.

Use this API for CRUD operations on *templates* and *zones*. This API also assists with retrieving and rendering templates for particular navigation states.

To learn how to use templates in the UI, see *Use Predictive Merchandiser Templates*.

	Use `/api/` before the templating operation, particularly if you are using this API for load testing. For example, use `/api/templating/render/APP_NAME` instead of `/templating/render/APP_NAME`. This ensures the API call does not go through the Fusion UI, which can slow down your response time.

Front-end Development

Ultimately, Fusion is the back end for your own search applications.

- Your application uses Fusion's REST API to interact with the Fusion system. The REST API supports all the features available in the Fusion UI. At a minimum, your application uses the `/query/QUERY_PROFILE_ID?SEARCH_QUERY` *endpoint* to query Fusion collections.
- *Recommendations* are a way to use aggregations to enhance the search experience. Based on the current search, or signals collected previously, Fusion can return results that are relevant in the end user's current context.
- Certain front-end features require some Fusion configuration:
  - *Autocomplete*
  - *Faceting*
  - *Stopwords*
  - *Synonyms*

Autocomplete

Autocomplete, also known as auto-suggest, look-ahead, or type-ahead, is a feature that displays a list of common search terms that begin with, or are contained by, the query string. For example, the query "search technology" might result in an autocomplete list with results "search technology rocks", "i love search technology", and/or "this is a search technology company", depending on how the autocomplete has been configured. Like most other search components, Fusion leverages Solr's autosuggest *(https://solr.apache.org/guide/suggester.html)* component. Assuming auto-suggest is configured in the Solr config, it is sufficient to enable it in Fusion in order to use the component.

REST API Examples for Front-End Development

This document provides examples of queries and calls that may be useful for front-end development of your application.

Prerequisites

Before working with the Fusion API, ensure that you have specified API *permissions* or *roles* to allow access to the API.

Simple queries

Send signals to Fusion from front-end applications

The Signals API stores signals as JSON for indexing into a signals collection. See *Signals API* for more information.

REQUEST

```
https://FUSION_HOST:FUSION_PORT/api/signals/COLLECTION_NAME
```

ENDPOINTS

POST	
/signals/COLLECTION_NAME	Send one or more signals for indexing

PARAMETERS

`commit`	`optional`	The `commit` parameter can potentially cause resource problems if your application has a high signal volume, or if it sends all signals as individual POST requests in real-time.
`collection`	`required`	Collection ID
`params`	`optional`	Configuration parameters For more information, see *Parallel Bulk Loader*.

EXAMPLE

```
curl \ -u USERNAME:PASSWORD -X POST -H 'Content-type:application/json' -d @- \
https://FUSION_HOST:FUSION_PORT/api/signals/docs?commit=true \ <<EOF [{ "params": {
"query": "Televisiones Panasonic 50 pulgadas", "filterQueries": ["cat00000",
"abcat0100000", "abcat0101000", "abcat0101001"], "docId": "2125233" }, "type":
"click", "timestamp": "2011-09-01T23:44:52.53Z" }, { "params": { "query": "Sharp",
"filterQueries": ["cat00000", "abcat0100000", "abcat0101000", "abcat0101001"],
"docId": "2009324" }, "type": "click", "timestamp": "2011-09-05T12:25:37.42Z" } ]
EOF
```

Send queries to Fusion using the Query Profile

REQUEST

```
https://FUSION_HOST:FUSION_PORT/api/apps/APP_NAME/query/QUERY_PROFILE?QUERY
```

ENDPOINTS

POST	
/signals/COLLECTION_NAME	Send one or more signals for indexing

PARAMETERS

`rows`	`optional`	The number of documents (results) to return
`wt`	`required`	The response format. For example, XML, JSON, CSV
`fl`	`optional`	The list of fields to return for each document in the response

EXAMPLE

```
https://FUSION_HOST:FUSION_PORT/api/apps/Movies/query/Movies_main?
q=star&fq=genres_ss:(“Sci-Fi”)&rows=5
```

Complex Queries

Call the query profiles assigned to a zone

The Render API is part of the *Templating API*. It is the main way to ingest the content of templates and their contained zones, which will further drive the search experience. Templates/zones data cannot be incorporated with the *Query Profile* or *Query Pipelines* APIs.

You can create templates in the Fusion UI, specifying trigger criteria for each template, such as search terms or filters. You can then add zones to these templates, choosing a query profile to be queried for each zone along with other configurable data. When you query the Render API, Fusion determines which template’s trigger criteria is satisfied by the query and calls the query profiles designated by the zones in the triggered template, passing the original query on to each query profile. The results from each query profile are returned alongside the zone configuration in the final response.

For example, a web front-end can consume this API to drive a search results page. It will inspect the list of returned zones, using each zone’s configuration to decide how to display each query profile result in the UI. This allows you to manage the search experience with the Fusion UI, using the template trigger criteria and zone configuration along with different query profiles to define what content should be shown based on the search context.

Templates and zones have both a "staging" and "live" (or "published") version. The staging parameter present on all Render API endpoints determines the version of the content that will be used. Modifications to the templating content only show in the "staging"

view until you publish the content.

The Render API works similarly to the Query API in that it supports all of the same ways of providing a query, and the responses for each zone are the same structure. However, there are four differences:

- You do not need to specify a query profile in your request
- The endpoint paths are different
- The Render API determines which template to trigger based on the query terms, filters, and the templates' configurations
- The provided query is issued against one or more query profiles (one for each zone in the triggered template), and the response from each query profile is included in the Render API response.

See *Query Profile API* for more information.

REQUEST

```
https://FUSION_HOST:FUSION_PORT/api/templating/render/APP_NAME
```

See *Templating API* for more supported parameters.

ENDPOINTS

GET	
/render/APP_NAME	Retrieve and render the template for the given navigation state. By default will only trigger published versions of templates, specify staging=true to instead trigger staging versions of templates.
/renderId/TEMPLATE_ID	Retrieve and render the template with the specified id. All provided query params will be passed on to each query profile invoked during the rendering.
POST	
/render/APP_NAME	Retrieve and render the template for the given navigation state. By default will only trigger published versions of templates, specify staging=true to instead trigger staging versions of templates.
/renderDSL/APP_NAME	Retrieve and render the template for the given navigation state. By default will only trigger published versions of templates, specify staging=true to instead trigger staging versions of templates. The provided DSL request object + any additional request parameters are passed through to each query profile when rendering each zone.
/renderId/TEMPLATE_ID	Retrieve and render the template with the specified id. All provided query params will be passed on to each query profile invoked during rendering.
/renderIdDSL/TEMPLATE_ID	Retrieve and render the template with the specified id, passing the provided DSL request to each query profile invoked during rendering.

EXAMPLE

The example query specifies `some query` as the query terms and is restricted to only return templates of the `landing` type. It will retrieve the template with trigger criteria that is satisfied by the query terms `some query`.

By specifying `staging=true`, it will only consider the current "staging" version of the templating content, which are templates/zones that haven't been published yet. Omitting the staging param would return only the published versions of templates/zones.

```
GET https://FUSION_HOST:FUSION_PORT/api/templating/render/APP_NAME?
q=QUERY&templateTypes=landing&staging=true
```

Queries for Recommendations

Create a personalized list of potential interests

The Recommendations API uses signals to generate a personalized list of interests. See *Recommendations API* for more information.

REQUEST

```
https://FUSION_HOST:FUSION_PORT/api/recommend/APP_NAME/{endpoint}?q=QUERY
```

ENDPOINTS

GET	
itemsForQuery	Get items for a defined query. The response is a list of *document IDs* and their weights.
queriesForItem	Get queries for a defined item (a document ID). This finds the top queries that led users to the defined item. The response is a list of query terms and their weights.

EXAMPLES

```
curl -u USERNAME:PASSWORD
https://FUSION_HOST:FUSION_PORT/api/recommend/lucidworks102/itemsForQuery?
q=ipod&fq=count_d:4&debug=true
```

```
curl -u USERNAME:PASSWORD
https://FUSION_HOST:FUSION_PORT/api/recommend/lucidworks102/queriesForItem?
docId=9225439`
```

Connectors SDK

Fusion comes with a wide variety of connectors, but you can also develop custom Fusion connectors using the Connectors SDK.

To get started, clone the public repository at `https://github.com/lucidworks/connectors-sdk-resources/` *(https://github.com/lucidworks/connectors-sdk-resources/)*.

- See *Java Connector Development* to learn about developing a Java-based connector.
- See *Develop A Custom Connector* for step-by-step instructions.

Unresolved directive in <stdin> - include::https://raw.githubusercontent.com/lucidworks/connectors-sdk-resources/{sdk-version}/README.asciidoc[lines=1..-1]

	For information about Fusion 4.2.x, see *Fusion 4.2.x Connectors SDK*.

Checkpoints in the Connectors SDK

Use Cases

Incremental Re-crawl

Incremental re-crawl can be supported when a Changes API is available (e.g., Jive, Salesforce, OneDrive). When a Changes API is available, it's necessary to provide an input parameter to be tracked, such as a date, link, page token, or other. The input parameter is generated (retrieved) while running the first job. During the next job, that parameter will be used to query the Changes API and retrieve new, modified, and deleted objects.

The SDK provides a way to store the input parameters, establishing **checkpoints**, and use them in the subsequent jobs.

Checkpoint Design

Fetcher implementations can emit checkpoint messages by calling:

```
fetchContext.emitCheckpoint(CHECKPOINT)_ID, checkpointMetadata);
```

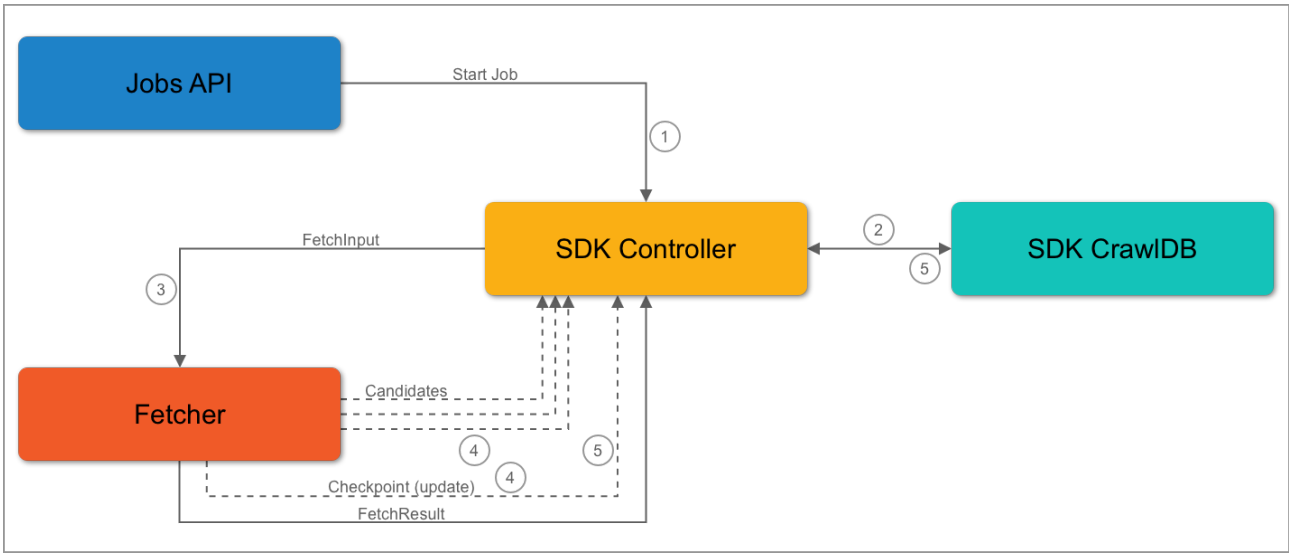
After the checkpoint is emitted, Fusion will handle this message as follows:

- The checkpoint will be stored in the CrawlDB with the appropriate status.
- The checkpoint will not be used in the current job.

In subsequent job runs, Fusion will check the CrawlDB for any previously stored checkpoints. If any are available, only those checkpoints will be sent to the fetchers; no other input types will be sent. If checkpoints are not available, all other items in the CrawlDB (Documents, FetchInputs, Errors, etc.) will be sent to the fetchers instead.

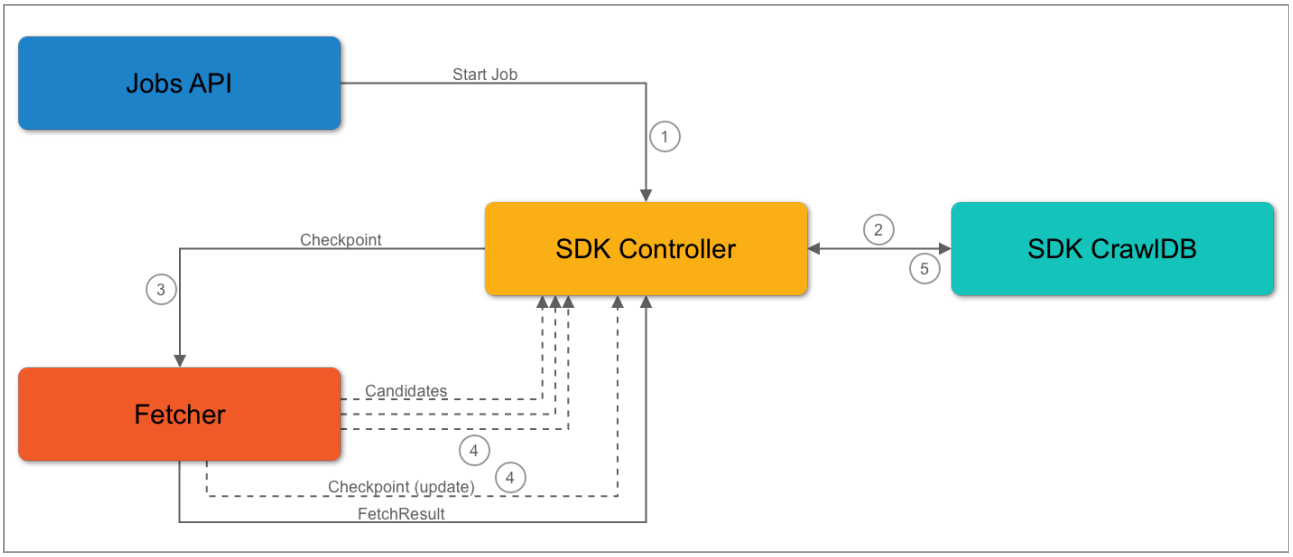
	In order to update a checkpoint, it must be emitted using its original ID. The ID is the only way the SDK controller can identify and update a checkpoint.

First Job Flow



1. The Jobs API sends a start job request to the SDK controller.
2. The SDK controller queries the SDK CrawlDB to check for items.
3. It's the first job, so the SDK CrawlDB is empty. The controller will send the initial FetchInput to the fetcher.
    a. During the job, the fetcher receives a FetchInput.
4. The fetcher can then emit candidates and/or checkpoints.
5. When the SDK controller receives a checkpoint message, the checkpoint is stored or updated in CrawlDB. It will also process the other items it has received.
    a. The SDK controller will not send the checkpoint to the fetcher in the same job.

Second Job Flow



1. The Jobs API sends a start job request to the SDK controller.
2. The SDK controller queries the SDK CrawlDB to check for items.
3. It's the second job, so checkpoints are stored in the SDK CrawlDB. The controller will send the checkpoints to the fetcher.
4. The fetcher receives and detects the checkpoints. Then, the fetcher emits candidates and updates the checkpoint. The update may take place at a later point, but the checkout *must* be updated.
    a. If the checkpoint data matches current data, the fetcher will emit the same checkpoint.
5. The SDK controller will process the candidates and update the checkpoint data in the SDK CrawlDB.

Stopping a Running Job

When a job is stopped, the current state of the job is stored so that it can be completed when the job is resumed.

Stop Handling Design

The SDK controller will keep track of the incomplete/complete items during a job. An incomplete item is an item that was emitted as a candidate but has not been sent to the fetcher to be processed. Alternatively, the fetch implementation may not have emitted the FetchResult message for that item. The incomplete item is stored in the SDK CrawlDB and marked as incomplete.

A completed item is one that was emitted first as a candidate and also sent to the fetcher to be processed. The fetcher completes the process by sending back a FetchResult. This item is then marked as complete by the SDK controller by setting the `blockID` field in the item metadata to match the `blockID` of the current job.

A `blockId` is used to quickly identify items in the CrawlDB which may not have been fully processed, or completed. A completed job is one that naturally stops due to source data being fully processed, as opposed to jobs that are manually stopped or fail.

`blockId's identify a series of one or more jobs. The lifetime of a `blockId` spans from either the start of the initial crawl (or immediately after a completed one), all the way to completion. The SDK controller will generate and use a new `blockId` when:

- The current job is the first job.
- The previous job's state is `FINISHED`.

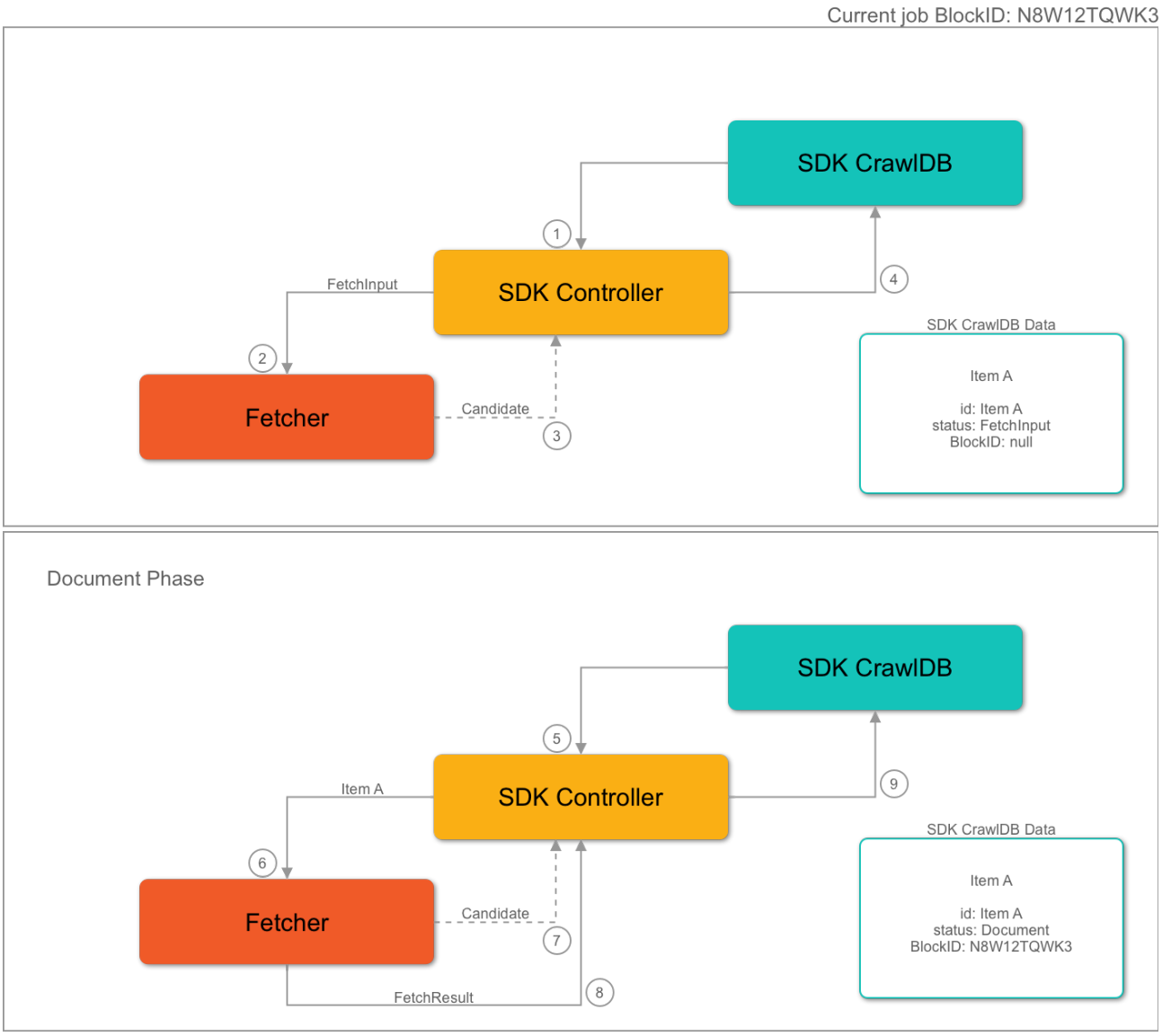
When a job starts and the previous job did not complete, the previous job's state is `STOPPED`. In this case, the previous job's blockId is reused. The same blockId will be reused until the crawl successfully completes. The SDK controller will continue checking the CrawlDB for incomplete items, which are identified by having a blockID that doesn't match the previous job blockID. This approach ensures all items within the job are completed before the next job beings, even if the job was stopped multiple times before completion.

When an item is considered a new candidate, the item's blockId does not change. Later, when the item is fully processed by the fetcher, the blockId is added to the item metadata and stored in the SDK CrawlDB. The item is then considered complete but will only be sent to fetchers when a new blockId is generated.

When all items are complete, the SDK will check for checkpoints, as detailed in *Checkpoint Design*.

	If there are incomplete items from the previous job stored in the SDK CrawlDB, ***only those items*** will be processed during the next job.

Item Completion Flow



1. The SDK controller gets a `FetchInput` from the SDK CrawlDB and sends it to the fetcher.
2. The fetcher receives the FetchInput.
3. The fetcher emits a candidate: `Item A`.
4. The controller receives the candidate and stores it in the SDK CrawlDB. Mandatory fields are set to the item metadata, but the `blockId` field is not set.
5. Later, in the same job, the candidate Item A is selected by the SDK controller, which sends it to a fetcher.
6. The fetcher receives the candidate and processes it.
7. The fetcher emits a `Document` from the candidate.
8. The fetcher emits a `FetchResult` to the SDK controller.
9. The SDK controller receives both the Document and the FetchResult.
    a. If processing the Document, the item status is updated to Document in SDK CrawlDB.
    b. If processing the FetchResult, the item's blockId is set to the current job's blockId.

Transient candidates

Some connector plugins require that a new job start from the latest checkpoints *and not* attempt to reprocess incomplete candidates. For that purpose, emit those candidates with `Transient=true`. The `IncrementalContentFetcher` is an example.

Java Connector Development

Fusion 5.4 and later

Lucidworks Connector Java SDK

Overview

The Lucidworks Connector SDK provides a set of components to support building all types of Fusion connectors. Connectors can be created by implementing some of the classes described in this document and using the provided build tool.

Connectors can be deployed directly within a Fusion cluster or outside Fusion as a remote client. To deploy a connector into Fusion, upload the build artifact (as a zip file) to the Fusion blob API. To deploy remotely, you can use the Fusion Connector Plugin Client *(https://github.com/lucidworks/connectors-sdk-resources/blob/v4.0.0/java-sdk/plugin-client.asciidoc)*.

Examples

The example connectors are a great place to start understanding how to build a Java SDK Connector. Have a look at the example connectors documentation *(https://github.com/lucidworks/connectors-sdk-resources/blob/v4.0.0/java-sdk/connectors/README.md)*.

Security

Read more about security for Java SDK connectors here *(https://github.com/lucidworks/connectors-sdk-resources/blob/v4.0.0/java-sdk/security.asciidoc)*.

Java SDK Framework Overview

Only a few components must be implemented to compile and build a functioning plugin. Here, we give an overview of each of these primary components and related concepts. We start by describing a minimal project directory layout, followed by details on the components.

Project Layout

This documentation and examples use Gradle for building connectors. Other build tools can be used instead. Check out the Gradle quickstart *(https://docs.gradle.org/current/userguide/tutorial_java_projects.html)* for Java Projects.

The minimal project layout is as follows:

```
my-connector
├── build.gradle
├── gradle.properties
├── settings.gradle
└── src
    └── main
        └── java
            └── Plugin.java
            └── Config.java
            └── Fetcher.java
```

build.gradle

This is a typical Gradle build file. A few important settings are required. You can have a look at an example here *(https://github.com/lucidworks/connectors-sdk-resources/blob/v4.0.0/java-sdk/connectors/build.gradle)*.

gradle.properties

The build utility used for packaging up a connector depends on this file and contains required metadata about the connector, including its name, version, and the version of Fusion it can connect to. Have a look at the example here *(https://github.com/lucidworks/connectors-sdk-resources/blob/v4.0.0/java-sdk/connectors/simple-connector/gradle.properties)*.

src/main/java

This is where Java code lives by default. This location can be changed, but in this documentation we use the Gradle default.

The Plugin

The `Plugin` class describes the various subcomponents of a connector, as well as its dependencies. The current set of subcomponents include the `Fetcher`, validation, security filtering, and configuration suggestions.

You can think of the `Plugin` class as the component that glues everything together for an implementation. It is also a place where fetch 'phases' are defined, by binding fetcher classes to phase names.

You can see an example here *(https://github.com/lucidworks/connectors-sdk-resources/blob/v4.0.0/java-sdk/connectors/simple-connector/src/main/java/com/lucidworks/connector/plugins/random/RandomContentPlugin.java)*. Notice that each subcomponent can have a set of unique dependencies, or several subcomponents can share dependencies.

The Configuration

The configuration component is an interface, which extends a base `Model` interface. By simply adding methods with `@Property` annotations, you can dynamically generate a type-safe configuration object from a connector's configuration data. Similarly, you can generate a Fusion-compatible JSON schema by implementing `Model`.

You can see an example of a configuration implementation here *(https://github.com/lucidworks/connectors-sdk-resources/blob/v4.0.0/java-sdk/connectors/simple-connector/src/main/java/com/lucidworks/connector/plugins/random/config/RandomContentConfig.java)*.

More detailed information on the configuration and schema capabilities can be found here *(https://github.com/lucidworks/connectors-sdk-resources/blob/v4.0.0/java-sdk/configuration.asciidoc)*.

The Fetcher

The `Fetcher` is where most of the work is done for any connector. This interface provides methods that define how data is fetched and indexed.

For a connector to have its content indexed into Fusion, it must emit messages, which are simple objects that contain metadata and content. See the *message definitions* below for more details.

Lifecycle

The `Fetcher` has a simple lifecycle for dealing with fetching content. The flow outline is as follows:

```
start() – called once for each fetcher when the Job is started fetch() – called for each input to fetch stop() – called once for each fetcher when the Job is stopped
```

start()

When a `ConnectorJob` starts, the Fetcher's `start()` method is called. The main use case for a start() call is to run setup code. This method is called on every fetcher bound to a phase once per job run.

fetch()

This is the primary fetch method for a crawler-like connector (for example, a file system or web connector). Messages emitted here can include `Documents`, which are indexed directly into the associated content collection. There are several other types of messages; see *Message Definitions*.

stop()

Called once per job run, at the end, on every fetcher bound to a phase.

Phases

Connectors can define "phases", which are distinct blocks of fetch processing. This makes it possible to break up fetching into steps, so that (for example), the first phase may fetch only metadata, while the second fetches content etc..

Phases are defined in the 'Plugin' class, by binding a fetcher class to a phase name. A single fetcher class can be bound to multiple phases.

For each phase, the `start`, `fetch`, and `stop` methods are called on the fetcher instance bound to the phase.

See the example here *(https://github.com/lucidworks/connectors-sdk-resources/blob/v4.0.0/java-sdk/connectors/simple-connector/src/main/java/com/lucidworks/connector/plugins/random/RandomContentPlugin.java)*.

Message Type Definitions

Candidate

A `Candidate` is metadata emitted from a `Fetcher` that represents a resource to *eventually* fetch. Once this message is received by the controller service, it is persisted, then added to a fetch queue. When this item is then dequeued, a connector instance within the cluster is selected, and the message is sent as a `FetchInput`. The `FetchInput` is received in the `fetch()` method of the connector. At this point, the connector normally emits a `Content` or `Document` message, which is then indexed into Fusion.

The general flow of how Candidates are processed is the key to enabling distributed fetching within the connectors framework.

FetchInput

The `FetchInput` represents an item to be fetched. Example values of `FetchInput` are a file path, a URL, or a SQL statement. `FetchInputs` are passed to the fetch() method of a Fetcher and are derived from `Candidate` metadata.

Document

A `Document` is a value that is emitted from a connector and represents structured content. Once the controller service receives a `Document` message, its metadata is persisted in the crawl-db and then sent to the associated `IndexPipeline`.

Content

A `Content` message represents raw content that must be parsed in order to be indexed into Fusion. They are analogous to InputStreams, and their bytes are streamed to Fusion. `Content` types are actually composed of three different subtypes: * ContentStart - this tells Fusion that a stream of raw content is coming. It includes the `content-type` and any other metadata related to the source data. * ContentItem - this is a message that contains one smaller chunk of the raw content. These messages are streamed sequentially from a connector to Fusion, and these are feed (without explicit buffering) directly to the index pipeline. * ContentStop - this messages indicates to Fusion that the `Content` is done. The end result of sending a `Content` stream, is a set of parsed documents within the Fusion Collection associated with the connector.

Skip

`Skip` messages represent items that were not fetched for some reason. For example, items that fail validation rules related to path depth or availability. Each Skip can contains details on why the item was skipped.

Error

`Error` messages indicate errors for a specific item. For example, when a connector's `fetch()` method is called with an nonexistent `FetchInput`, the connector can emit an error that captures the details ("not found", for example).

Errors are recorded in the data store, but are not sent to the associated `IndexPipeline`.

Delete

`Deletes` tell the controller service to remove a specific item from the data store and associated Fusion collection.

AccessControlItem

`AccessControlItem` message represent group, user, role, etc. used for security filtering.

Item state transitions

Initial Item state	Valid transition state	Comment
Candidate	FetchInput	Candidates emitted are stored as FetchInput
FetchInput	Document	
	Skip	
	Error	
	AccessControlItem	
Document	Skip	
	Error	
Skip	Document	
	Error	
Error	Document	
	Skip	
Checkpoint	Checkpoint	Checkpoint should not transition to any other state
Delete	No transition state	
AccessControlItem	No transition state	

Understanding when and when not to make a Candidate "transient"

The term `transient` refers to a Candidate's persistance with respect to the Crawl-db.

- When an emit candidate is `transient=true`, this means that we will clear the candidate from the Crawl-db after each crawl completes. Transient candidates will not be re-fetched in the next crawl. This means that subsequent crawls will need to create new candidates using the data source properties and checkpoints. An example of when you would want to do this is with a connector that has an "delta change" feature that can provide you the Created/Updated/Removed documents since you last crawled. You can avoid having to revisit every candidate from previous crawls because you have the means to know exactly what was changed. This is much faster than revisiting each candidate in the entire crawl database… so you should always prefer this option when it is a possibility.
- When an emit candidate is `transient=false`, this means that we want to store the candidate in the CrawlDB, then we will send them to be fetched again so that they are reevaluated in each subsequent crawl. An example of when you want to do this is in a "Re-crawl Strategy" where you must revisit an item previously crawled explicitly each time subsequent crawls are run. Because revisiting each item is typically quite slow, you would only do this when the data source you are crawling provides no "delta change" feature that can provide you the Created/Updated/Removed documents since you last crawled.

Developing a Connector

The simplest way to get started developing a connector is to review Simple Connector *(https://github.com/lucidworks/connectors-sdk-resources/tree/v4.0.0/java-sdk/connectors/simple-connector/README.asciidoc)*.

Dependencies

At a minimum, all plugins require the connector SDK dependency. And more than likely, a handful of other dependencies as well. Examples would be an HTTP client, a special client for connecting to a third-party REST API, or maybe a parser library that can handle parsing video files.

Specify these dependencies with the Java build tool of your choice (for example, Gradle or Maven). But getting them into your code and even instantiating them is partly handled by the connector SDK.

Dependency Injection

Allowing these plugins to specify runtime dependencies makes them simpler to unit test, and generally more flexible. When defining a connector plugin, the object used for the definition also supports adding bindings for these dependencies. A general guideline to follow when determining if something should be injected or not:

1. Is it desirable to have multiple implementations of the component?
2. Is the component a third-party library that has either difficult to control side effects?

The connectors use the Guice *(https://github.com/google/guice)* framework for dependency injection and can be directly used when defining a connector plugin.

Simple Demo Connector

This connector generates a configurable number of documents, all with random titles and body fields.

Quick start

1. Clone the repo:

```
git clone https://github.com/lucidworks/connectors-sdk-resources.git
cd connectors-sdk-resources/java-sdk/connectors/
./gradlew assemblePlugins
```

2. This produces one zip file, named `simple-connector.zip`, located in the `build/plugins` directory. This artifact is now ready to be uploaded directly to Fusion as a connector plugin.
3. See the following *instructions* on how to build, deploy, and run the plugin.

Connector properties

Random content generator properties

Property Name	Property description
Total	The total number of documents to generate
Minimum number of sentences	The minimum number of sentences to generate per document, the random generator will use this value as lower bound to calculate a random number of sentences
maximum number of sentences	The maximum number of sentences to generate per document, the random generator will use this value as upper bound to calculate a random number of sentences

How to use the connector

- Create a configuration with the properties listed above.
- After the first job is completed, the connector will index the same number of documents as defined in the `Random content generator properties.Total` property.

Connectors SDK Javadocs

See the Javadocs for your version of the Connectors SDK:

- *SDK 1.4 Javadocs*
- *SDK 1.5 Javadocs*
- *SDK 2.0 Javadocs*
- *SDK 3.0 Javadocs*
- *SDK 4.0 Javadocs*

Index Stage SDK

Overview

Lucidworks provides an Index Stage SDK in a public repository on GitHub *(https://github.com/lucidworks/index-stage-sdk)* with all the resources you need to develop custom index stages with Java.

Clone the repository to get started:

```
git clone https://github.com/lucidworks/index-stage-sdk
```

See Gradle quickstart *(https://docs.gradle.org/current/userguide/tutorial_java_projects.html)* documentation for more information on Java Projects.

Concepts

Index stage configuration

The index stage configuration file *(https://github.com/lucidworks/index-stage-sdk/blob/master/index-stage-plugin-sdk/src/main/java/com/lucidworks/indexing/config/IndexStageConfig.java)* defines configuration options specific to the index stage instance. The options defined in this configuration file are available to the user in the Fusion UI and the API. The plugin configuration class extends the index stage configuration file and is annotated with `@RootScheme`.

Adding `@Property` and type annotations to your stage configuration interface methods defines metadata and type requirements for your plugin configuration fields. This is similar to Fusion's *connector configuration schema*.

APIs

The Index Stage SDK includes several APIs for communication with other Fusion components via the Fusion *(https://github.com/lucidworks/index-stage-sdk/blob/master/index-stage-plugin-sdk/src/main/java/com/lucidworks/indexing/api/fusion/Fusion.java)* object. This object is passed to the stage during initialization.

RestCall
: The RestCall API *(https://github.com/lucidworks/index-stage-sdk/blob/master/index-stage-plugin-sdk/src/main/java/com/lucidworks/indexing/api/fusion/RestCall.java)* provides access to the *Fusion REST API*. You can find an example *(https://github.com/lucidworks/index-stage-sdk/blob/master/examples/sample-plugin-stage/src/main/java/com/lucidworks/sample/query/ExternalQueryStage.java)* of its usage in the Index Stage SDK repository.

Blobs
: The Blobs API *(https://github.com/lucidworks/index-stage-sdk/blob/master/index-stage-plugin-sdk/src/main/java/com/lucidworks/indexing/api/fusion/Blobs.java)* enables interactions with the *Blob Store API*.

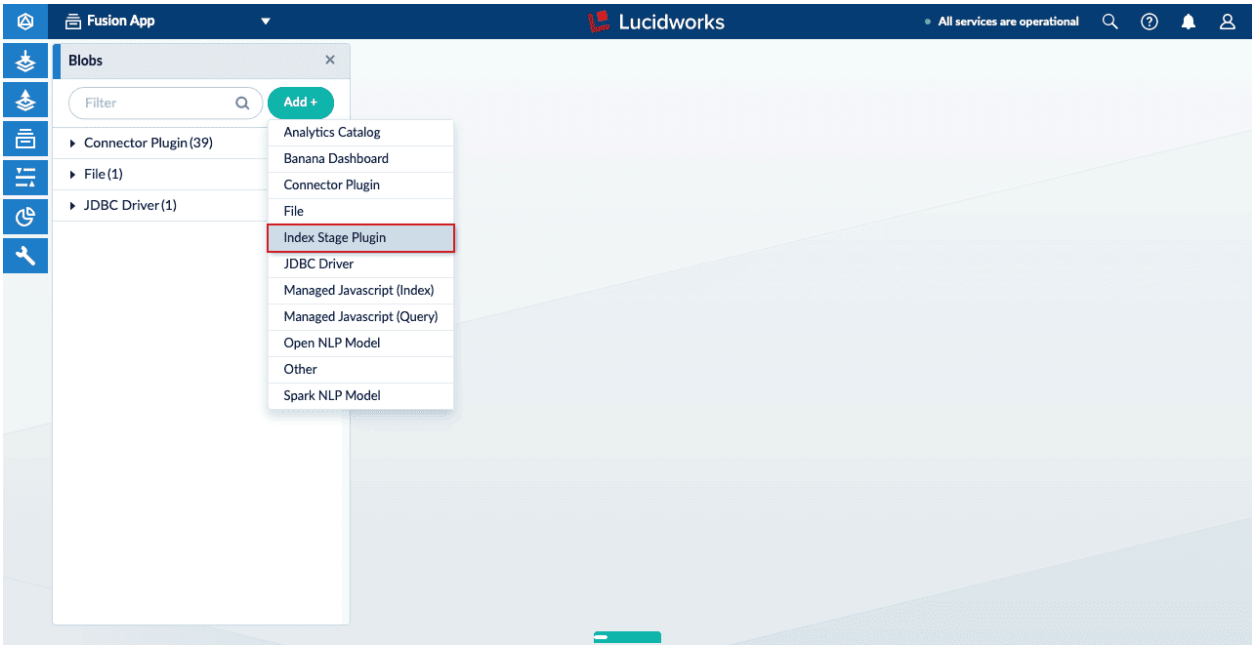
Documents
: The Documents API *(https://github.com/lucidworks/index-stage-sdk/blob/master/index-stage-plugin-sdk/src/main/java/com/lucidworks/indexing/api/fusion/Documents.java)* provides a method for creating new document instances. This is useful for custom stages that *output multiple documents* from a single input documentation.

Plugins

A plugin is a `.zip` file that contains one or more index stage implementations. The file contains `.jar` files for stage definitions and additional dependencies. It also contains a manifest file that holds the metadata Fusion uses to run the plugin.

Plugins are uploaded to the Blob store:

1. Navigate to **System** > **Blobs**.
2. Click **Add**.
3. Select **Index Stage Plugin**.



4. Click **Browse…** and select your plugin file.
5. Click **Upload**.

Plugin stage classes must implement the `com.lucidworks.indexing.api.IndexStage` *(https://github.com/lucidworks/index-stage-sdk/blob/master/index-stage-plugin-sdk/src/main/java/com/lucidworks/indexing/api/IndexStage.java)* interface and be annotated with `com.lucidworks.indexing.api.Stage` *(https://github.com/lucidworks/index-stage-sdk/blob/master/index-stage-plugin-sdk/src/main/java/com/lucidworks/indexing/api/Stage.java)* annotation. For additional convenience, stage implementation can extend the `com.lucidworks.indexing.api.IndexStageBase` *(https://github.com/lucidworks/index-stage-sdk/blob/master/index-stage-plugin-sdk/src/main/java/com/lucidworks/indexing/api/IndexStageBase.java)* class, which already contains initialization logic and some helpful methods.

Lifecycle

Creation and initialization

Fusion begins by creating an `IndexStage` instance. After the index stage is created, it is initialized using the `init(T config, Fusion fusion)` method. This allows for the creation of internal storage instructions and the validation of the configuration.

Initialization occurs immediately after the stage configuration is saved in Fusion. The stage can be maintained and used by Fusion for extensive periods of time, even if no documents are being processed through the stage. This should be considered when making decisions on resource allocation.

Document processing

Once the initalization process completes, Fusion calls the `process` method for each document the index pipeline processes.

In most use cases, index stages process a single input document and emit a single output document. For these cases, the `process(Document document, Context context)` method should be used.

In other cases, index stages process a single input document but emit multiple output documents. For these cases, the `process(Document document, Context context, Consumer<Document> output)` method should be used. The output documents are sent by calling `output.accept(doc)`.

A single stage instance can be used to process multiple documents, and the `process` method can be called from multiple concurrently running threads. Additionally, Fusion can initialize and maintain multiple stage instances with the same configuration in separate indexing service nodes. Therefore, it's important to ensure the plugin stage implementation is thread-safe and the processing logic is stateless.

	If the index stage throws an exception while processing a document, that document will not be processed further. It does *not* prevent other documents from being processed. Check the logs for information regarding the exception.

Logging

The Index Stage SDK uses the SLF4J Reporter logging API.

Index Stage SDK Javadocs

See the *Javadocs* for complete details about all packages and classes.

Pipeline Javadocs

See the *5.4 pipeline Javadocs*.

Fusion App Studio

App Studio

App Studio is a collection of software and resources that helps you *build* modern, user-friendly apps to search and visualize your Fusion data. This includes:

- ***Appkit*** - The framework that drives App Studio Enterprise.
- ***App Studio Enterprise*** - The standalone web application which you connect to your Fusion collection.

App Studio requires a running instance of *Fusion*, or access to one of our *supported datasources*, with one or more collections of indexed data.

Looking to get started?

Appkit

The Appkit framework provides a library of search application components encapsulated as simple building blocks that combine to create complex applications for search and information retrieval.

See the *reference docs* for more information about the Appkit framework.

Modules

Appkit's suite of search components is modular, which means you are able to get a working solution into place quickly using tried-and-tested components.

Read more about modules.

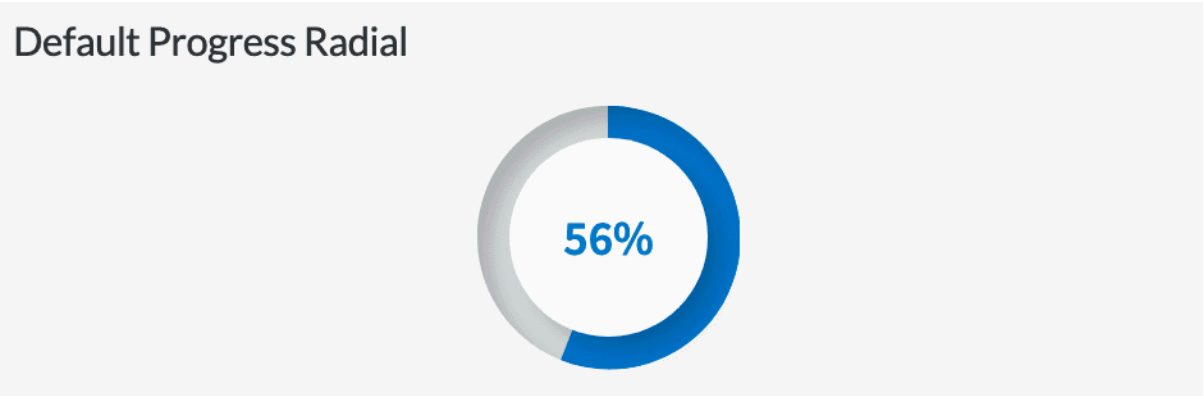
Tags

The App Studio markup language defines a set of HTML-like elements, or *tags*, for defining search queries to the backend and visualizing the search results and facets, handling all the complexities of connecting to Fusion.

For example, adding a radial progress bar is as simple including the following tag:

```
<progress:radial percentage="56"> </progress:radial>
```

This results in the following:



Over 150 tags are provided in the Appkit framework.

Read more about tags.

Services

App Studio exposes a number of frontend JavaScript services which encapsulate reusable processes. These services are based on the AngularJS framework. There are also backend Java web services that perform the same functions as many of the frontend services,

including platform search endpoints, platform series endpoint, query suggestion/typeahead services, csv/excel export, and facet search.

Read more about services.

Search Platforms

Appkit integrates with a number of search engines, data warehouses and web service endpoints, even surfacing information from multiple sources simultaneously. The platform abstraction covers both search and storage sides, meaning that Appkit will implement for each platform the necessary protocols to save or index content for any platform. A number of platform adapters are available out-of-the-box with Appkit, and *adding new ones* is a simple exercise.

Read more about search platforms.

Concepts

- *Appkit Concepts*
- *Appkit Configuration*
- *Extending Appkit*

Reference

- *App Studio Versions FAQ*
- *Accessing Data*
- *App Deployment*
- *FAQ*

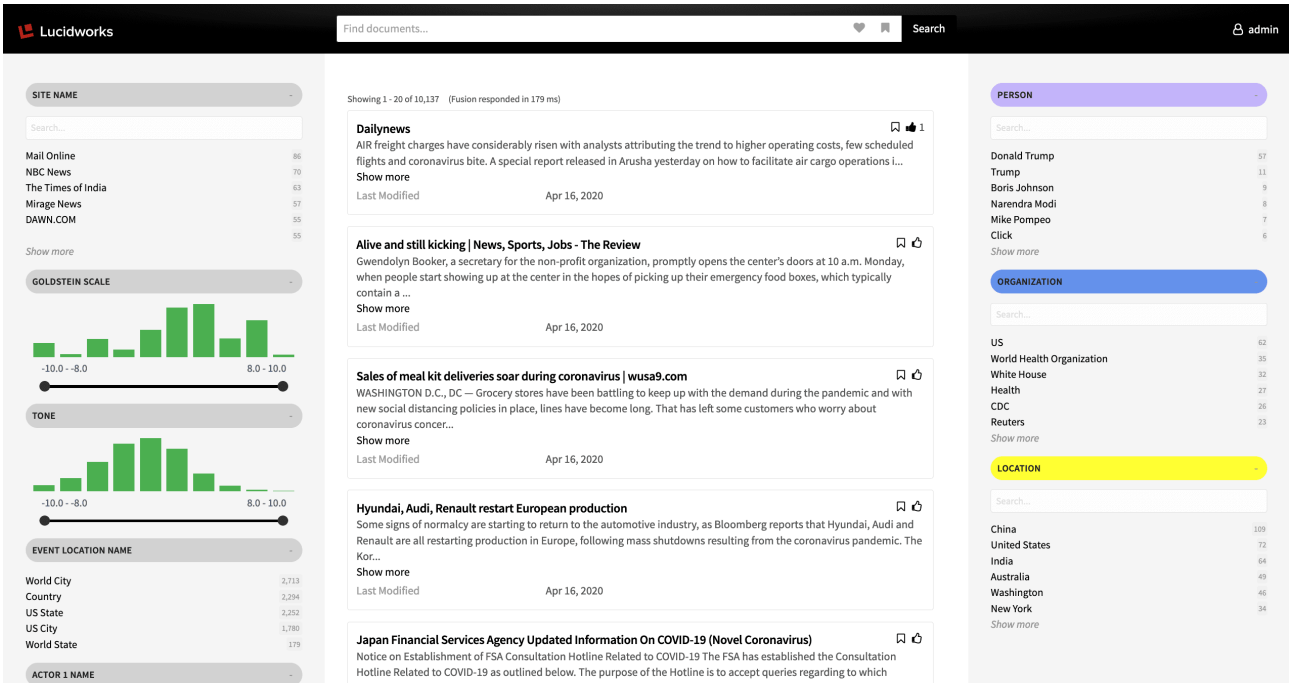
How-to

- *Upgrade Appkit in Existing Apps*

App Studio Enterprise

App Studio Enterprise is an Appkit based web application that connects your Fusion instance, or other *datasources*, to the *Appkit* design framework. It comes preloaded with the Appkit framework library and a starter search app template. This allows you to develop and preview your search application using your data before deploying it to a production environment.

Getting started only takes a few minutes and is as simple as *downloading App Studio Enterprise*, configuring the application to use your Fusion instance, and launching. See the README file included in the download for full instructions.



Concepts

- *App Studio Enterprise Concepts*
- *App Studio Enterprise Downloads*

- *Getting Started with App Studio Enterprise*

Reference

- *App Studio Versions FAQ*
- *FAQ*

How-to

- *Building from Source*
- *Deploy App Studio Enterprise*

Versions FAQ

How are these products versioned?

- **Appkit** is versioned according to updates to the code base.
- **App Studio Enterprise** versions are loosely tied to the version of the Appkit framework it includes. An update from **4.8.0** to **4.9.0** may not include any changes to App Studio Enterprise.

What about compatibility?

- Any version of **Appkit** can be used with Fusion 4.x. However, only Appkit 4.6.0+ can be used with Fusion 5.y.
- **App Studio Enterprise** can be used with any version of Fusion or Appkit.

Can these products be updated independently?

- **Appkit** can be *updated* independently, assuming there are no version *compatibility* issues.
- **App Studio Enterprise** is released when a new version of Appkit is released. As a result, App Studio Enterprise itself does not need to be updated. To move to the latest code base, simply update the Appkit version within the project.

See the *FAQ section* for more frequently asked questions.

App Studio Reference Guides

The reference section contains information that is presented without explanation of the general concepts. It includes content such as definitions and attributes for services, tags, modules, etc.

For conceptual information, see the *concepts section*.

- *Accessing Data*
  The articles in this section describe key aspects of accessing data.
- *Deployment Overview*
  This article describes general app deployment with App Studio.
- *FAQ*
  The articles in this section answer frequently asked questions about various aspects and functionality of App Studio.
- *Filters*
  Filters are AngularJS based filters that can be implemented in the markup by piping the filter name to a data object, which transforms the object. See *Filter Syntax* for more information.
- *Gallery*
  The gallery provides visual examples of App Studio elements, with code snippets that you can copy, paste, and edit in your own App Studio views.
- *Installation*
  This section contains various articles on the installation, system requirements, and supported environments of App Studio.
- *Licensing*
  This article contains basic licensing details.
- *Modules*
  Appkit's suite of search components is modular, which means you are able to get a working solution into place quickly using tried-and-tested components.
- *Services Reference*
  App Studio exposes a number of frontend JavaScript services which encapsulate reusable processes. These services are based on the AngularJS framework. There are also backend Java web services that perform the same functions as many of the frontend services, including platform search endpoints, platform series endpoint, query suggestion/typeahead services, csv/excel export, and facet search.
- *Tags*
  The App Studio markup language defines a set of HTML-like elements, or *tags*, for defining search queries to the backend and visualizing the search results and facets, handling all the complexities of connecting to Fusion.

Services

App Studio exposes a number of frontend JavaScript services which encapsulate reusable processes. These services are based on the AngularJS framework. There are also backend Java web services that perform the same functions as many of the frontend services, including platform search endpoints, platform series endpoint, query suggestion/typeahead services, csv/excel export, and facet search.

$filterService

Description

Utility for parsing and creating filter objects.

Methods

- **parse(filterString)**

  Method to parse URL representations of a filter.

  Parameters

Param	Type	Details
filterString	string	filter to parse.

  Example

  $filterService.parse('foo["bar"]');

$lightningLog

Description

Twigkit factory for logging.

Methods

- **debug(file, message)**

  Method to debug

  Parameters

Param	Type	Details
file	string	The file the message is coming from
message	string	The message to be displayed

  Example

  $lightningLog.debug(instance,message);

- **error(file, message)**

  Method to return error message

  Parameters

Param	Type	Details
file	string	The file the message is coming from
message	string	The message to be displayed

  Example

  $lightningLog.error(instance,message);

- **warn(file, message)**

  Method to return warn message

  Parameters

Param	Type	Details
file	string	The file the message is coming from
message	string	The message to be displayed

  Example

  $lightningLog.warn(instance,message);

$lightningUrl

Description

Twigkit factory for getting and setting the location (url).

Methods

- **addFilterObjectToUrl(filter, reloadSearch)**

  Method to add a filter to the url

  Parameters

Param	Type	Details
filter	Object	The filter object to add to the url.
reloadSearch *(optional)*	boolean	Whether to reload search results.

  Example

  $lightningUrl.addFilterObjectToUrl(filter);

- **addFilterToLocationCopy(fieldName, actual, display, orFilter, location, exclude, match)**

- **addFilterToUrl(fieldName, actual, display, reloadSearch, orFilter, exclude, match)**

  Method to add a filter to the url

  Parameters

Param	Type	Details
fieldName	string	The name of the filter to apply.
actual	string	The actual value of the filter.
display	string	The display value of the filter.
reloadSearch *(optional)*	boolean	Whether to reload search results.
orFilter *(optional)*	string	If the filter is to be ORed
exclude *(optional)*	boolean	If the filter is to be exclude values
match *(optional)*	string	'left_partial' or 'partial' or 'right_partial' or 'exact'

  Example

  $lightningUrl.addFilterToUrl(filterName, actual, display, reloadSearch);

- **addRangeFilterDisplayToUrl(fieldName, from, to, displayFrom, displayTo, reloadSearch, orFilter, fromExcluded, toExcluded, exclude)**

  Method to add a filter to the url

Parameters

Param	Type	Details
fieldName	string	The name of the filter to apply.
from	string	The from value of the filter.
to	string	The to value of the filter.
displayFrom	string	The from display value of the filter.
displayTo	string	The to display value of the filter.
reloadSearch *(optional)*	boolean	Whether to reload search results.
orFilter *(optional)*	boolean	If the filter is to be ORed
fromExcluded *(optional)*	string	whether the from is to be excluded.
toExcluded *(optional)*	string	Whether the to is to be excluded.
exclude *(optional)*	boolean	Whether the values should be excluded.

Example

```
$lightningUrl.addRangeFilterDisplayToUrl(fieldName, from, to, display, reloadSearch, orFilter, fromExcluded, toExcluded) ;
```

- **addRangeFilterToUrl(fieldName, from, to, displayFrom, displayTo, reloadSearch, orFilter, toLabel, fromExcluded, toExcluded, exclude)**

  Method to add a filter to the url

Parameters

Param	Type	Details
fieldName	string	The name of the filter to apply.
from	string	The from value of the filter.
to	string	The to value of the filter.
displayFrom	string	The from display value of the filter.
displayTo	string	The to display value of the filter.
reloadSearch *(optional)*	boolean	Whether to reload search results.
orFilter *(optional)*	boolean	If the filter is to be ORed
toLabel *(optional)*	string	The label which appears between the from and to values.
fromExcluded *(optional)*	string	whether the from is to be excluded.
toExcluded *(optional)*	string	Whether the to is to be excluded.
exclude *(optional)*	boolean	Whether the values should be excluded.

Example

$lightningUrl.addRangeFilterToUrl(fieldName, from, to, displayFrom, displayTo, reloadSearch, orFilter, toLabel, fromExcluded, toExcluded);

- **createFilter(fieldName, actual, display, orFilter, exclude, match, type)**

  Creates a filter for the URL

Parameters

Param	Type	Details
fieldName	string	The name of the filter to apply.
actual	string	The actual value of the filter.
display	string	The display value of the filter.
orFilter *(optional)*	string	If the filter is to be ORed (field or value)
exclude *(optional)*	boolean	If the filter is to be exclude values
match *(optional)*	string	'left_partial' or 'partial' or 'right_partial' or 'exact'
type *(optional)*	string	Explicitly set the type of the filter, 'string' or 'number'. A string filter value will be wrapped in double quotes.

Returns

string	Returns a filter that can be added to the URL

- **createFilterFromObject(filter)**

  Creates a filter for the URL

  Parameters

Param	Type	Details
filter	Object	The filter object or string to create a filter from.

  Returns

string	Returns a filter that can be added to the URL

- **createRangeFilter(fieldName, from, to, displayFrom, displayTo, orFilter, toLabel, fromExcluded, toExcluded, exclude)**

  Creates a range filter for the URL

  Parameters

Param	Type	Details
fieldName	string	The name of the filter to apply.
from	string	The from value of the filter.
to	string	The to value of the filter.
displayFrom	string	The from display value of the filter.
displayTo	string	The to display value of the filter.
orFilter *(optional)*	string	If the filter is to be ORed (value | field)
toLabel *(optional)*	string	The label which appears between the from and to values.
fromExcluded *(optional)*	string	whether the from is to be excluded.
toExcluded *(optional)*	string	Whether the to is to be excluded.
exclude *(optional)*	boolean	Whether the values should be excluded.

  Returns

string	Returns a filter that can be added to the URL

- **createRangeFilterDisplay(filterName, from, to, display, orFilter, fromExcluded, toExcluded, exclude)**

  Creates a range filter for the URL

Parameters

Param	Type	Details
filterName	string	The name of the filter to apply.
from	string	The from value of the filter.
to	string	The to value of the filter.
display	string	The display value of the filter.
orFilter *(optional)*	string	If the filter is to be ORed (value | field)
fromExcluded *(optional)*	string	whether the from is to be excluded.
toExcluded *(optional)*	string	Whether the to is to be excluded.
exclude *(optional)*	boolean	Whether the values should be excluded.

Returns

string	Returns a filter that can be added to the URL

- **getAllUrlParameters()**

  Method to get a copy of all URL parameters

  Returns

object	Object literal of all URL parameters

  Example

  $lightningUrl.getAllUrlParameters();

- **getFiltersFromUrl(fieldName)**

  Method to gets all of a set of filters from the url

  Parameters

Param	Type	Details
fieldName	string	Only get filters on the specified field.

  Returns

Array	Returns values of the filterName from URL.

  Example

  $lightningUrl.getFiltersFromUrl(filterName);

- **getUrlParameter(parameter, defaultValue)**

  Method to get a url parameter

Parameters

Param	Type	Details
parameter	string	The parameter to get from the URL.
defaultValue *(optional)*	string	The value to return if parameter returns null.

Returns

var	Returns value of parameter from URL or defaultValue.

Example

$lightningUrl.getUrlParameter(parameter, defaultValue);

- **getUrlParametersAsArray(parameter, defaultValue)**

  Method to gets an array of url parameters

  Parameters

Param	Type	Details
parameter	string	The parameter to get from the URL.
defaultValue *(optional)*	string	The value to return if parameter returns null.

  Returns

Array	Returns values of the parameter from URL or defaultValue in an array.

  Example

  $lightningUrl.getUrlParametersAsArray(parameter, defaultValue);

- **removeFiltersFromUrl(filterName, reloadSearch)**

  Method to remove all of a set of filters from the url

  Parameters

Param	Type	Details
filterName	string	The name of the filter to remove.
reloadSearch *(optional)*	boolean	Whether to reload search results.

  Example

  $lightningUrl.removeFiltersFromUrl(filterName, reloadSearch);

- **removeFiltersFromUrl(filterName, reloadSearch)**

  Method to remove all of a set of filters from the url

Parameters

Param	Type	Details
filterName	string	The name of the filter to remove.
reloadSearch *(optional)*	boolean	Whether to reload search results.

Example

```
$lightningUrl.removeFiltersFromUrl(filterName, reloadSearch);
```

- **resetAllParameters(reloadSearch)**

  Resets all URL Parameters

  Parameters

Param	Type	Details
reloadSearch	boolean	Whether to reload the search

- **resetParameters(parameters, reloadSearch)**

  Resets a selection of parameter types

  Parameters

Param	Type	Details
parameters	array	types to reset
reloadSearch	boolean	Whether to reload the search

- **setUrlParameter(parameter, value, reloadSearch)**

  Method to set a url parameter

  Parameters

Param	Type	Details
parameter	string	The parameter to add to the URL.
value	Arraystring	The value to assign to the parameter.
reloadSearch *(optional)*	boolean	Whether to reload the results.

  Example

```
$lightningUrl.setUrlParameter(parameter, value, reloadSearch);
```

- **setUrlParameters(query, reloadSearch)**

  Method to set a group of url parameters from a query.

Parameters

Param	Type	Details
query	Object	Query object from which URL Parameters are to be used.
reloadSearch *(optional)*	boolean	Whether to reload the search once all the parameters have been set.

Example

$lightningUrl.setUrlParameters(query);

- **urlContainsFilter(filterName, actual, display, orFilter, exclude, match)**

  Method to set if filter is in the url

  Parameters

Param	Type	Details
filterName	string	The name of the filter to apply.
actual	string	The actual value of the filter.
display	string	The display value of the filter.
orFilter *(optional)*	string	If the filter is to be ORed on `value` or `field`
exclude *(optional)*	boolean	If the filter is to be exclude values
match *(optional)*	string	'left_partial' or 'partial' or 'right_partial' or 'exact'

  Returns

boolean	Whether the URL contains the filter.

- **urlContainsFilterFromObject(filter)**

  Method to see if filter is in the url

  Parameters

Param	Type	Details
filter	object	The filter object

  Returns

boolean	Whether the URL contains the filter.

$twigkit

Description

Twigkit factory with methods for accessing the Twigkit API.

Methods

- **cancelAllResponses()**

  Fires an event to cancel the current response being requested from the backend.

  Example

  $twigkit.cancelAllResponses();

- **deleteBookmark(id)**

  Sends a request to delete a bookmark to the API.

  Parameters

Param	Type	Details
id	string	The id of the bookmark to delete.

  Returns

HttpPromise	Returns a promise which will resolve depending on the response from the API.

  Example

  $twigkit.deleteBookmark(id);

- **deleteComment(platform, id)**

  Sends a request to delete a comment to the API.

  Parameters

Param	Type	Details
platform	Object	The platform object (retrievable from *Platform Service*).
id	string	The id of the comment to delete.

  Returns

HttpPromise	Returns a promise which will resolve depending on the response from the API.

  Example

  $twigkit.deleteComment(platform, id);

- **deleteLike(platform, id)**

  Sends a request to delete a like to the API.

Parameters

Param	Type	Details
platform	Object	The platform object (retrievable from *Platform Service*).
id	string	The id of the bookmark to delete.

Returns

HttpPromise	Returns a promise which will resolve depending on the response from the API.

Example

$twigkit.deleteLike(platform,id);

- **deleteSavedQuery(id)**

  Sends a request to delete a saved query to the API.

  Parameters

Param	Type	Details
id	string	The id of the saved query to delete.

  Returns

HttpPromise	Returns a promise which will resolve depending on the response from the API.

  Example

  $twigkit.deleteSavedQuery(id);

- **deleteTopic(id)**

  Sends a request to delete a topic to the API.

  Parameters

Param	Type	Details
id	string	The id of the topic to delete.

  Returns

HttpPromise	Returns a promise which will resolve depending on the response from the API.

  Example

  $twigkit.deleteTopic(id);

- **getComments(platform, target, collection, topic)**

  Sends a request to get comments from the API.

Parameters

Param	Type	Details
platform	Object	The platform object (retrievable from *Platform Service*).
target *(optional)*	string	Returns all comments that apply to the specified target document.
collection *(optional)*	string	
topic *(optional)*	string	

Returns

HttpPromise	Returns a promise which will resolve depending on the response from the API.

Example

$twigkit.getComments(platform, target);

- **getConstant(defaultValue)**

  Gets a constant declared to the lightning module.
  A constant can be added to your application like so

```
angular .module('lightning').constant('contextPath', 'My Constant Value');
```

  You can then get this value in your controller and apply it to the scope or rootScope (if applied to the rootScope you can use `{{$root.myConstant}}` in your views.

```
angular.module('twigkitLightApp') .controller('MainCtrl', ['$scope',
'$stateParams','$twigkit', function ($scope, $stateParams,$twigkit) { $scope.params
= $stateParams; $scope.test123 = $twigkit.getConstant('test','No 123 Found Here');
$rootScope.test = $twigkit.getConstant('test',''); }]);
```

  Parameters

Param	Type	Details
defaultValue	string	Value to be returned if context path does not exist.

  Returns

string	Returns the constant value.

  Example

  $twigkit.getConstant(defaultValue);

- **getContextPath(defaultValue)**

  Gets the context path if it is available.

Parameters

Param	Type	Details
defaultValue *(optional)*	string	Value to be returned if context path does not exist. Defaults to '/'

Returns

string	Returns a string of the context path or the default value.

Example

```
$twigkit.getContextPath(defaultValue);
```

- **getGraphFacet(query, platform, source-field, target-field, measure, measure-field)**

  Gets graph data for a facet

Parameters

Param	Type	Details
query	string	The query to use against the platform.
platform	string	The name of the platform to use.
source-field	string	Name of a primary field to facet on (for example "Country"). The values of this facet will define the set of graph nodes with outgoing edges.
target-field	string	Name of a secondary field to facet on (for example "City"), for every facet value of the primary field. The values of this facet will define the set of graph nodes with incoming edges. For example, say the primary facet (source-field) has values for Country=Germany and Country=France and the secondary facet (target-field) has values for City=Berlin, City=Munich and City=Paris. Then we add graph edges Germany-->Berlin, Germany-->Munich and France-->Paris.
measure *(optional)*	string	Name of a numerical measure that defines the weight of each graph edge. When omitted, this defaults to the counts of the facets defined by the target-field. Some possible measures include "sum", "mean" and "min". (Only applicable if not using `data` attribute).
measure-field *(optional)*	string	Field to measure, when the "measure" option is specified.

Returns

HttpPromise	Returns a promise which will resolve depending on the response from the API.

Example

```
$twigkit.getGraphFacet('query', 'platform', 'facet');
```

- **getLikes(options)**

  Sends a request to get a likes from the API.

  Parameters

Param	Type	Details
options	Object	Object containing options to be sent in the get request. Like target, collection, topic etc.

  Returns

HttpPromise	Returns a promise which will resolve depending on the response from the API.

  Example

  $twigkit.getLikes(options);

- **getUser()**

  Calls the Twigkit API to return the current user
  To enable this you need to make sure that your security.conf file contains the line `webservice-enabled: true`
  Your security.conf file can be found in 'resources/conf/security/security.conf'.
  For spring security you will need the following rule to allow access to the user api, if parts of the site are public.
  `<intercept-url pattern="/twigkit/api/user" access="isAnonymous() or isAuthenticated()"/>`

  Returns

HttpPromise	Returns a promise which will resolve depending on the response from the API.

  Example

  $twigkit.getUser();

- **onCancelAllResponses(callback, scope)**

  When the cancel response event is fired the callback function will be called.

  Parameters

Param	Type	Details
callback	Function	The function to be called when the cancel event is fired.
scope	Object	The `$scope` object for unsubscribing to the event when the scope is destroyed.

  Example

```
$twigkit.onCancelAllResponses(function () { console.log('Responses have been cancelled'); }, $scope); $twigkit.cancelAllResponses();
```

- **postLike(platform, id, target, topic, collection)**

  Sends a request to create a like to the API.

Parameters

Param	Type	Details
platform	Object	The platform object (retrievable from *Platform Service*).
id	string	The id of the bookmark to delete.
target	string	A unique identifier for a document that is being bookmarked. Example values: document123 Maximum length: 2083 characters
topic *(optional)*	number	A topic to which the new bookmark should belong.
collection *(optional)*	string	The collection to which the target document belongs.

Returns

HttpPromise	Returns a promise which will resolve depending on the response from the API.

Example

$twigkit.postLike(platform, id, target, topic, collection);

- **search(searchQuery, platform, params, ignore-loading-bar, canceller)**

  Search method sends a query to a Twigkit Platform endpoint and gets a result.

Parameters

Param	Type	Details
searchQuery	string	The search term to search against the results.
platform	Object	The platform object (retrievable from *Platform Service*).
params *(optional)*	Object	Other parameters to pass to the API.
ignore-loading-bar *(optional)*	boolean	Whether the loading bar across the top of the page should be included for this http request. Default:true
canceller *(optional)*	Promise	A promise object (`$q.defer()`) used to cancel the request by using `canceller.promise()`

Returns

HttpPromise	Returns a promise which will resolve depending on the response from the API.

Example

$twigkit.search(query, platform, params);

- **searchFacet(facet, platform, query, rpp, facetQuery, removeFilters, facetField, wildcard, canceller)**

  Calls the Twigkit API to return filters from a specific facet based on the searchQuery.

Parameters

Param	Type	Details
facet	string	The name of the facet to search.
platform	Objectstring	The platform object (retrievable from *Platform Service*) or the name of the platform.
query	Object	The query object.
rpp	Number	The number of filters to be returned by the API.
facetQuery	string	The facet query to use against the API end point.
removeFilters *(optional)*	boolean	Whether to remove the current filters for the facet when performing the search.
facetField *(optional)*	string	If provided this field may be used to create a filter on the facet which may not be the same as the facet name itself.
wildcard *(optional)*	string	Pass the wildcard settings for the request, either left, right, both or none.
canceller *(optional)*	Promise	A promise object (`$q.defer()`) used to cancel the request by using `canceller.promise()`

Returns

HttpPromise	Returns a promise which will resolve depending on the response from the API.

Example

$twigkit.searchFacet(facet,platform,query,facetQuery);

- **searchSeriesFacet(query, platform, facet, measure, rpp)**

  Gets series data for a facet.

  Parameters

Param	Type	Details
query	string	The query to use against the platform.
platform	Object	The platform object (retrievable from *Platform Service*).
facet	Object	The facet to search upon.
measure	string	What fact to use as an alternative to count as quantitative value for a filter.
rpp	Number	The number of results to fetch.

  Returns

HttpPromise	Returns a promise which will resolve depending on the response from the API.

  Example

  $twigkit.searchSeriesFacet(query, platform, facet);

- **searchSeriesFields(query, platform, fields, rpp)**

  Gets series data for a result list.

Parameters

Param	Type	Details
query	string	The query to use against the platform.
platform	Object	The platform object (retrievable from *Platform Service*).
fields	Object	The facet to fields upon.
rpp	Number	The number of results to fetch.

Returns

HttpPromise	Returns a promise which will resolve depending on the response from the API.

Example

```
$twigkit.searchSeriesFacet(query, platform, fields);
```

$util

Description

Twigkit factory with utility functions for Development.

Methods

- **hasChildren(transclude)**

  Method to check if the element has children.

  Parameters

Param	Type	Details
transclude	Object	The transclude function from the elements post or pre linking function.

  Returns

boolean	True if element has children.

  Example

```
$util.hasChildren(transclude);
```

BookmarkService

Description

Twigkit service storing bookmarks over a session

CollaborationService

Description

Appkit service for accessing Appkit Collaboration API's

Methods

- **delete(annotation, platform, id)**

  Deletes an annotation

  Parameters

Param	Type	Details
annotation	string	The type of collaboration annotation to delete. Valid values are:- **comment** - For deleting a comment. **bookmark** - For deleting a bookmark. **topic** - For deleting a topic. **link** - For deleting a like. **saved-query** - For deleting a saved-query.
platform	string	The name of the platform.
id	string	The ID of the annotation to be deleted.

  Returns

HttpPromise	Returns a promise which will resolve depending on the response from the API.

  Example

  CollaborationService.delete('comment', 'platform', 123456);

- **getCommentAddedEventName(target, topic, collection)**

  Method for returning the event emitted when a comment is added

  Parameters

Param	Type	Details
target	string	The comment target.
topic *(optional)*	string	The topic associated to the comment.
collection *(optional)*	string	The collection associated to the comment.

  Returns

string	The event name that will be emitted.

  Example

  CollaborationService.getCommentAddedEventName(target, collection, topic);

- **post(annotation, platform, data)**

  Creates an annotation

Parameters

Param	Type	Details
annotation	string	The type of collaboration annotation to delete. Valid values are:- **comment** - For creating a comment. **bookmark** - For creating a bookmark. **topic** - For creating a topic. **link** - For creating a like. **saved-query** - For creating a saved-query.
platform	string	The name of the platform.
data	object	The data object for creating the required annotation.

Returns

HttpPromise	Returns a promise which will resolve depending on the response from the API.

Example

```
CollaborationService.post('comment', 'platform', data);
```

- **query(annotation, platform, target, collection, topic, results)**

  Gets an array of Collaboration Annotations (comments, likes, topics or bookmarks) based on the options provided.

  Parameters

Param	Type	Details
annotation	string	The type of collaboration annotation you are searching for. Valid values are:- **comments** - Returns a list of comments **bookmarks** - Returns a list of bookmarks **topics** - Returns a list of topics **links** - Returns a list of likes. **saved-queries** - Returns a list of saved-queries.
platform	string	The name of the platform.
target *(optional)*	string	When provided it will return only annotations associated with this target (not applicable to topics).
collection *(optional)*	string	When provided it will return only annotations associated with this collection (not applicable to topics).
topic *(optional)*	string	When provided it will return only annotations associated with this topic (not applicable to topics).
results *(optional)*	number	The number of annotations to return.

  Returns

HttpPromise	Returns a promise which will resolve depending on the response from the API.

  Example

```
CollaborationService.query('comments', 'platform', 'foo-bar');
```

DateUtils

Description

Twigkit service for useful date functions

Methods

- **dateSortResults(results, dateField, oldestFirst)**

  Sort results array by date

  Parameters

Param	Type	Details
results	mixed	Array of results from response
dateField	string	Result field to use for ordering
oldestFirst	boolean	If true the results will be order from oldest to newest

  Returns

mixed	Sorted array of results

- **groupByDateRanges(results, dateField, groupBy)**

  Group results into date ranges, used for the timeline

  Parameters

Param	Type	Details
results	mixed	Sorted array of results
dateField	string	Result field to use for ordering
groupBy	string	Optionally specify how to group the results, by default it tries to get the best fit

  Returns

mixed	Array of grouped results

FieldService

Description

Twigkit service for getting the values from a field

Methods

- **getDisplayValue(field)**

  Gets the correct property from the value object

  Parameters

Param	Type	Details
field	mixed	The field to find a display value in.

- **getValues(field)**

  Gets an array of all values for the field

  Parameters

Param	Type	Details
field	mixed	The field to find all the values in

- **isRange(value)**

  Returns a boolean based on if the value (field.val) is a range.

  Parameters

Param	Type	Details
value	object	The val object from a field or filter to check if it is a range.

ModalService

Description

Twigkit service for registering and un-registering modals

Methods

- **close(name)**

  Hides the modal in the view.

  Parameters

Param	Type	Details
name	string	The ID of the modal to remove.

  Example

  ModalService.close(name);

- **onClose(name, callback, scope)**

  When the modal's close event is fired the callback function will be called.

  Parameters

Param	Type	Details
name	string	The name of the modal, if null the callback will fire whenever a modal is closed
callback	Function	The function to be called when the cancel event is fired.
scope	Object	The `$scope` object for unsubscribing to the event when the scope is destroyed.

  Example

  ```
  ModalService.onClose('MyModal', function (modalId) { console.log('Modal: 'modalId + ' has been closed'); }, $scope); ModalService.close('my_modal');
  ```

- **register(modal)**

  Registers the modal to the factory

  Parameters

Param	Type	Details
modal	string	The modal object.

  Example

  ModalService.register(modal);

- **show(name)**

  Shows the modal in the view.

Parameters

Param	Type	Details
name	string	The ID of the modal to remove.

Example

ModalService.show(name);

- **unregister(name)**

  Un-registers the modal removing it from the view and the factory.

  Parameters

Param	Type	Details
name	string	The ID of the modal to remove.

  Example

  ModalService.unregister(name);

PlatformService

Description

Twigkit service for setting and getting Platforms.

QueryIdService

Description

Twigkit service for setting and getting query references for tracking.

QueryService

Description

Twigkit service for queries

Methods

- **clearQueries()**

  Will remove all the queries from the factory.

  Example

  QueryService.clearQueries();

- **convertHttpParamsToQuery(httpParams)**

  Converts http params to query value..

  Parameters

Param	Type	Details
httpParams	object	The httpParams of the query.

  Example

  QueryService.convertHttpParamsToQuery(value);

- **convertHttpParamsToUrl(httpParams)**

  Converts http params into a URL.

  Parameters

Param	Type	Details
httpParams	object	A query object as http params

  Example

  QueryService.convertHttpParamsToUrl(query, true);

- **convertQueryToHttpParams(query, excludeHiddenFilters)**

  Converts an object based query to HTTP Params.

  Parameters

Param	Type	Details
query	object	Query to convert
excludeHiddenFilters *(optional)*	boolean	Whether to include hidden filters in the params.

  Returns

object	Returns the query object as http params.

Example

QueryService.convertQueryToHttpParams(query);

- **convertQueryToUrl(query, includeHiddenFilters)**

  Converts query object into a URL, encoding necessary values correctly and optionally removing hidden filters

  Parameters

Param	Type	Details
query	object	The query object.
includeHiddenFilters	boolean	Whether to include hidden filters in the URL

  Example

  QueryService.convertQueryToUrl(query, true);

- **convertSortObjectToParam()**

  Will convert sort object into an parameter.

  Example

  QueryService.convertSortObjectToParam(obj);

- **convertSortParamToObject()**

  Will convert sort parameter into an object.

  Example

  QueryService.convertSortParamToObject(param);

- **getQuery(name)**

  Gets the value of the query from the factory. If no query is found, the factory will return null.

  Parameters

Param	Type	Details
name	string	Name of the query to retrieve.

  Returns

object	Returns the query object, if no query object exists it will return null.

  Example

  QueryService.getQuery(name);

- **getQueryAsHttpParams(name, excludeHiddenFilters)**

  Gets the value of the query from the factory. If no query is found, the factory will return null.

Parameters

Param	Type	Details
name	string	Name of the query to retrieve.
excludeHiddenFilters *(optional)*	boolean	Whether to include hidden filters in the params.

Returns

object	Returns the query object, if no query object exists it will return null.

Example

QueryService.getQueryAsHttpParams(name);

- **queryContainsFilterForField(query, fieldName, filter)**

  Will identify if the query contains the filter.

  Parameters

Param	Type	Details
query	object	The query object.
fieldName	string	The name of the field
filter	object	The filter object

  Example

  QueryService.queryContainsFilterForField(query, fieldName, filter);

- **removeFiltersForField(query, fieldName)**

  Will remove any filters relating to the field name

  Parameters

Param	Type	Details
query	object	The query object.
fieldName	string	The name of the field

  Example

  QueryService.removeFiltersForField(query, fieldName);

- **setQuery(name, value)**

  Adds the query value to the factory.

  Parameters

Param	Type	Details
name	string	Name to store the query under.
value	object	The value of the query.

Example

QueryService.setQuery(name,value);

Events

- **query_name_updated**

  When the query is set or changed Lightning will broadcast this message where name is the name of your query `query_name_updated`

  #### Type:

  broadcast

  #### Target:

  root scope

ResponseService

Description

Twigkit service for setting and getting responses.

Methods

- **clearResponses()**

  Removes all responses from the factory.

  Example

  ResponseService.clearResponses();

- **createResponseFromResults(results)**

  Creates a response object from the set of results you provide

  Parameters

Param	Type	Details
results	array	The list of results to add to this response.

  Returns

Object	Returns a newly created response object with the results provided.

  Example

  ResponseService.createResponseFromResults(results);

- **getResponse(name, keys)**

  Gets response from the factory.

  Parameters

Param	Type	Details
name	string	The name of the response to retrieve
keys *(optional)*	string	A comma separated list of keys you would like returned. If null whole response will be returned.

  Returns

Object	Returns the response object from the factory, if it doesn't exist it will return `null`

  Example

  ResponseService.getResponse(name);

- **setResponse(name, value)**

  Adds response to response factory for retrieval using the getResponse method. Broadcasts an event of 'responsename_updated' with the new response value.

Parameters

Param	Type	Details
name	string	The name of the response
value	Object	The response object to add to the factory.

Example

ResponseService.setResponse(name, value);

ScrollToService

Description

Appkit service providing functions for scrolling to the top or a particular element ID on the page.

Methods

- **scrollToElement(id)**

  Scrolls to a specific element based on ID (if no element id is provided it will scroll to the top of the page).

  Parameters

Param	Type	Details
id *(optional)*	string	The ID of the element to scroll to (if the ID is not defined then this function will scroll the user to the top of the page).

  Example

  ScrollToService.scrollToElement('foo');

TogglePanesService

Description

Twigkit service for the toggle pane(s) components

Methods

- **onTogglePane(panesId, callback, scope)**

  When the togglePanes event is fired the callback function will be called.

  Parameters

Param	Type	Details
panesId	string	The id of the toggle panes component where the callback is being fired.
callback	Function	The function to be called when the cancel event is fired.
scope	Object	The `$scope` object for unsubscribing to the event when the scope is destroyed.

  Example

```
TogglePanesService.onTogglePane('MyPaneId', function (paneNumber) {
console.log('Pane has been toggled to ' + paneNumber); }, $scope);
TogglePanesService.togglePane('MyPaneId', 1);
```

- **togglePane(panesId, paneNumber)**

  Fires an event to change the currently selected pane for a specific set of panes.

  Parameters

Param	Type	Details
panesId	string	The id of the toggle panes component to fire the toggle against.
paneNumber	number	The number of the pane to switch to (0 based so the first pane is 0, the second pane is 1 etc.).

  Example

  TogglePanesService.togglePane('MyPaneId', 1);

TranslationService

Description

Appkit Translation Service

Methods

- **addBundle(name, bundle)**

  Add a translation bundle with a specified name

  Parameters

Param	Type	Details
name	string	
bundle	object	

- **get(term, fallback, namespaces)**

  Get a translation for a term if it exists, otherwise return the original term

  Parameters

Param	Type	Details
term	string	The term to translate
fallback	string	The fallback to apply
namespaces *(optional)*	string	The bundle names to use. Comma delimited *(default: "default")*

  Returns

string	The translated term

- **put(key, translation, namespace, override)**

  Add a new translation

  Parameters

Param	Type	Details
key	string	
translation	string	
namespace *(optional)*	string	
override *(optional)*	string	

UserService

Description

Twigkit service for setting and getting User.

httpInterceptor

Description

Intercepts all `$http` requests that return with an error. It will then broadcast the event `httpInterceptorError` which will provide the error object containing the status code, status text and other error information.

Usage

```
angular module('myApp', ['lightning'] ) .* .run(['$rootScope', '$templateCache','$location','$timeout', function ($rootScope, $templateCache, $location,$timeout) { $rootScope.$on('httpInterceptorError', function(event,error) { //File not found is HTML if(error.config.headers.Accept == 'text/html'){ if(error.status == '404'){ alert('Page Not Found!'); //Timeout is required to redirect the page after all other events have fired $timeout(function(){ $location.path('search'); },0) } } });
```

Events

- **httpInterceptorError**

  Broadcast the event `httpInterceptorError` when a `$http` request receives an error, this will provide the error object containing the status code, status text and other error information.

  #### Usage

  ` angular module('myApp', ['lightning']) .* .run(['$rootScope', '$templateCache','$location','$timeout', function ($rootScope, $templateCache, $location,$timeout) { $rootScope.$on('httpInterceptorError', function(event,error) { //File not found is HTML if(error.config.headers.Accept == 'text/html'){ if(error.status == '404'){ alert('Page Not Found!'); //Timeout is required to redirect the page after all other events have fired $timeout(function(){ $location.path('search'); },0) } } });

  #### Type:

  broadcast

  #### Target:

  root scope

scrollAndResizeListener

Description

A Twigkit service that will fire an event when the window is scrolled or resized.

Methods

- **bindListener(listener)**

  A method for binding functions to a listener.

  Parameters

Param	Type	Details
listener	function	The function to run when the event occurs

  Returns

function	Runs the function when the event occurs.

  Example

  scrollAndResizeListener.bindListener(function(){ console.log('Window Scrolled or Resized'); });

Accessing Data

The articles in this section describe key aspects of accessing data.

- *Configuring the Search Platform*
  The Appkit search platform abstraction presents a unified way of carrying out common data operations such as searching, storing, and deleting information from the underlying repository. You can use multiple platforms on the same page to send a query to different repositories and aggregate the response. Each search platform is encapsulated as a `twigkit.platform.Platform` Java object.
- *Filter Syntax*
  This article describes the syntax of a single Appkit search filter to include in an HTTP request.
- *Formulating a Query*
  The Appkit Query tag (`<search:query>`) instantiates a search query that can be submitted to the underlying search platform.
- *Search Response*
  The Response tag (`<search:response>`) sends a Query to a given Platform and stores the Response in a JSP variable.

Configuring the Search Platform

The Generic Platform tag can instantiate all platforms by name or configuration.

What is an Appkit platform?

The Appkit search platform abstraction presents a unified way of carrying out common data operations such as searching, storing, and deleting information from the underlying repository. You can use multiple platforms on the same page to send a query to different repositories and aggregate the response. Each search platform is encapsulated as a `twigkit.platform.Platform` Java object.

Searching

When a platform is used for searching, it will map an Appkit query to something the underlying search engine will understand and send it off. When the search engine responds the platform will again map the response from the engine specific structure to a generic Appkit search response that can be processed by other Appkit components such as workflow processors and UI widgets.

Storing data

If a particular platform supports storing data, then it will take care of adapting an Appkit representation of a textual content to a piece of data that the underlying engine can store, and send it off for indexing or saving.

Usage

This example shows how to use any of the platforms using the generic Platform tag in an html page. This would retrieve a platform configuration from a file (under the `conf` directory).

```
<search:platform var="platform" conf="platforms.fusion.data"></search:platform>
```

Configuration parameters

`var (java.lang.String)`
Variable name to reference this `twigkit.model.Platform` instance.
Default: `platform`

`conf (java.lang.String)`
Retrieve a `twigkit.model.Platform` instance by either configuration name, or a platform annotation class name such as `twigkit.search.solr.Solr`. This is not the same as the platform class name used by the className attribute. Use either this or the name attribute.

`host (java.lang.String)`
The URI of the Solr Server. Must be specified using the full URI (for example, `http://localhost:8983/solr`; `localhost:8983` will not work).

For configuration settings for a specific provider, see *Search Platforms*.

Filter Syntax

This article describes the syntax of a single Appkit search filter to include in an HTTP request.

```
<filter-expression> ::= <field-expression> <group-expression> <value-expression
<display-expression> <field-expression> ::= <exclusion-expression> <field-optional-
expression> <field-name> <exclusion-expression> ::= "-" | "" <field-optional-
expression> ::= "~" | "" <group-expression> ::= "(" <group-number> ")" <group-
number> ::= <digit> | <digit> <group-number> <field-name> ::= <field-name-prefix>
<field-name-suffix> <field-name-prefix> ::= <letter> | "_" <field-name-suffix> ::=
<field-symbol> | <field-symbol> <field-name-suffix> <field-symbol> ::= <letter> |
<digit> | "_" | "-" | "." | ":" | " " | "" <value-expression> ::= <value-bracket-
left> <filter-value> <value-bracket-right> <value-bracket-left> ::= "[" | "("
<value-bracket-right> ::= "]" | ")" <filter-value> ::= <value> | <range-value>
<value> ::= <string-expression> | <numeric-value> | <date-value> <string-
expression> ::= <partial-match> <string-value> <partial-match> <string-value> ::=
'"' <text> '"' | "'" <text> "'" <partial-match> ::= '' | '*' <numeric-value> ::=
<floating-point> | <integer> <number-sequence> ::= <digit> | <number-sequence>
<digit> <integer> ::= "0" | <positive-digit> <number-sequence> <floating-point> ::=
<integer> | <integer> "." <number-sequence> <date-value> ::= <four-digits> "-"
<two-digits> "-" <two-digits> "'T'" <two-digits> ":" <two-digits> ":" <two-digits>
<two-digits> ::= <digit> <digit> <four-digits> ::= <two-digits> <two-digits>
<range-value> ::= <value> "," <value> | "," <value> | <value> "," <display-
expression-opt> ::= "" | "[" <display-expression> "]" <display-expression> ::= "["
'"' <display-value> '"' "]" | "[" "'" <display-value> "'" "]"
```

Where:

```
<letter> ::= "A" | "B" | "C" | "D" | "E" | "F" | "G" | "H" | "I" | "J" | "K" | "L"
| "M" | "N" | "O" | "P" | "Q" | "R" | "S" | "T" | "U" | "V" | "W" | "X" | "Y" | "Z"
| "a" | "b" | "c" | "d" | "e" | "f" | "g" | "h" | "i" | "j" | "k" | "l" | "m" | "n"
| "o" | "p" | "q" | "r" | "s" | "t" | "u" | "v" | "w" | "x" | "y" | "z" ;
<positive-digit> ::= "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9" <digit>
::= "0" | <positive-digit>
```

Here:

- `<group-number>` is an integer value denoting a grouping of filter constraints
- `<field-name>` is a the name of the field to apply the constraint to. A valid field name is one that matches the regular expression `[A-Za-z_][A-Za-z_0-9\-\.: ]`
- `<display-value>` is an optional label to attach to the filter
- `<filter-value>` is the actual constraint to be applied to the field named <field-name>

Formulating a Query

The Appkit Query tag (`<search:query>`) instantiates a search query that can be submitted to the underlying search platform.

Usage

This example shows how to create a Query using the Query tag. The Query tag can use all applicable parameters from the query string (HTTP request) to create a Query object.

As an example, if the URL of the page is `http://intranet/search?q=guidelines&rpp=10`, then the word 'guidelines' would be set as the query term, and the query would instruct the search engine to limit the results per page (rpp) to 10.

This example shows how you can override the `rpp` (short form of the `results-per-page` attribute) to fix how many results should be displayed per page, or override the value specified in the URL query string. To do this, add the long-form name of the attribute to override to the Query tag:

```
<search:query var="myQuery" parameters="*" results-per-page="5" />
```

The resulting query object can be accessed using an angular expression in the HTML page on which it is located. For example, `{{myQuery}}`.

You can also create a Query instance from another one using the from attribute for cloning the former:

```
<search:query var="query" from="otherQuery" results-per-page="5" />
```

Query Parameters

`var (java.lang.String)`
This is the name of the variable in which the query object will be stored.
Default: `query`

`parameters (java.lang.String)`
Parameters to use from the HTTP request to create the Query object. Set `parameters` to `*` to use all parameters. A comma-separated list of parameters to use is also valid.

`from (twigkit.model.Query)`
Create a new Query from existing Query object.

> Adding any of these query parameters means that the value is fixed for all queries, that is, the value overrides any parameters inherited from the URL query string.

`query (&q=) (java.lang.String)`
Use the query (or q) parameter to specify the query term to use. This will override default-query parameter of the Platform.

`modified-query (&mq=) (java.lang.String)`
Specify an alternative query that will be submitted to the search engine in place of the query parameter supplied by the user (but can include user submitted parameter). To the user it will appear as the query parameter had been used, but the modified-query will be submitted to the search engine.

`type (&t=) (java.lang.String)`
Specifies how to interpret multiple query terms. Takes one of three values: any (boolean OR the terms), all (boolean OR/AND the terms), or adv (interpret using search platform-specific advanced query language). For example, take the query apple ipod. With type="all" documents containing both the terms (for example, apple and ipod) will be returned, whereas with type="any" documents containing either term (for example, apple or ipod) will be returned. See special cases section below for type="adv" usage examples.
Default: `any`

`results-per-page (&rpp=) (java.lang.Integer)`
The number of results to display per page.
Default: 10

`max-results (&max=) (java.lang.Integer)`
The maximum number of results to evaluate.
Default: -1, which means no upper limit

`page (&p=) (java.lang.Integer)`
Set the page of results to display. Default values is 1 (1-based index) which would display the first page of results (1 to getResultsPerPage).
Default: 1

`facets (&fa=) (java.lang.String)`
Specifies which facets to evaluate and return. Specified as comma-separated list without spaces (for example, facets="cat,inStock"). Will override defaultFacets parameter of the Platform tag.

`fields (&fi=) (java.lang.String)`
A comma-separated list of fields that should be returned in the Response.

`sorts (&s=) (java.lang.String)`
Specifies fields and order for sorting. Sort order is specified by one of two characters: + for ascending and - for descending. To sort ascending by the field price, the sort would be specified as `sorts="+price"`.

Multilevel sorting is supported. To sort by multiple levels using the QueryTag use & to separate the values, for example, to sort by price ascending and then sku descending, use `sorts="+price&-sku"`. To do the same using the URL query string use multiple instances of the `s` parameter, for example, `?&s=+price&s=-sku`.

`view (&v=) (java.lang.String)`
Set which search engine view to use. Behavior will vary by installation.

`filters (&f=) (java.lang.String)`
Specifies filters to apply, specified as an ampersand (&) separated String. In the query string however there will be multiple `&f=` arguments. Filters are used to restrict field values in that appear in results. Also see *Alternative way to add Filters* below.

`facetsEnabled (java.lang.Boolean)`
Whether to calculate and return facets for the query. Behavior is platform dependent.

`custom (&c=) (java.lang.String)`
The custom attribute lets you send custom parameters to the underlying search platform. Parameters are specified using the format `parameter[value]` for a single parameter and corresponding value. Multiple parameters can be submitted with the QueryTag by separating them with & characters. For example:
`custom="parameter1[value1]&parameter2[value2]"`
To submit multiple custom parameters using the URL query string, use multiple instances of the `c` parameter. For example:
`?c=parameter1[value1]&c=parameter2[value2]`

Short format

Filters are specified using the format `fieldName[value]['display name']`. The `['display name']` part is optional in all cases.

For example, adding `filters="locations['san francisco']"` to the Query tag restricts the results to those with `san francisco` appearing in the locations field.

Multiple filters can be specified on the Query tag by using the separator characters &. For example, `filter="locations['san francisco']&people['obama']"` would restrict the results to those with the value `san francisco` appearing in the locations field and the value `obama` appearing the people field.

Multiple filters can be specified with the URL query string by adding multiple f parameters:

```
?f=locations['san francisco']&f=people['obama']
```

	If the value is a String it should be surrounded by quotes, all other primitives and dates should have the quotes omitted see below (URL query parameter examples).

For example:

- Filtering on String: `&f=stringField['My String']["String example"]`
- Filtering on Integer:`&f=integerField[123]["Integer example"]`
- Filtering on Double: `&f=doubleOrFloatField[123.45]["Double example"]`

Long format

When the query value of the filter differs from the display value the format is slightly different to accommodate this extra information: `filters="fieldName[value]["display name"]`.

Range format

Appkit supports range queries for numbers, dates and strings. For example `filters=salary[1000,100000]` will search the salary field for values between 1000 and 100000.

Date format

To filter by date (on date fields allowing range queries) use the DateFilter format (in all cases the display name is optional):

- To Filter on dates before the provided Date use: `filters=dateFieldName[,yyyy-MM-dd'T'HH:mm:ss]["Before the given date"]`
- To Filter on dates after the provided Date use: `filters=dateFieldName[yyyy-MM-dd'T'HH:mm:ss,]["After the given date"]`
- To Filter on dates within a given range use: `filters=dateFieldName[yyyy-MM-dd'T'HH:mm:ss,yyyy-MM-dd'T'HH:mm:ss]["Date range"]`
- From 2.10 onwards, to exclude the upper or lower limit of a date range it is possible to replace the `[` and `]` with a `(` and `)`. For example: `filters=dateFieldName[yyyy-MM-dd'T'HH:mm:ss,yyyy-MM-dd'T'HH:mm:ss)["Date range excluding upper limit"]`

Invariant filters

To configure a query with *invariant constraints*, you can add filter tags to the body of the query tag like so:

```
<search:query var="query" ... > <query:filter field="myField" value="myValue">
</query:filter> </search:query>
```

This constrains the query to all documents with the value "myValue" in "myField". It is not possible to use ranges with the `<query:filter>` tag. The value will always be interpreted as a string.

See the *`query:filter`* tag doc for available parameters.

Optional Operators

Appkit supports combining filters using different logical operators (connectives).

Enabling Logical Disjunction for Filters (OR connectives)

There are two ways of combining a set of filters using an inclusive disjunction where each filter represents an logically optional condition whilst keeping filters individually distinct. Each filter's optional state is represented as being either 'value', 'field' or 'none', with the corresponding Java Enumerations below.

'value' - Values within a given field

Filters for the same field that are 'value' optional will be combined using an OR operator. This group of filters will then be combined with the rest of the filters using a logical AND.

This would be the recommended approach when using checkboxes to select values within a facet. In that case the default behavior would be to allow the user to select multiple options which would be combined using an OR. This is encoded in the query URL using an asterisk after the filter value:

```
&f=foo['bar']&f=marx['groucho']*&f=marx['harpo']*
```

This would result in a query like: `(foo:bar AND (marx:groucho OR marx:harpo))`

'field' - Fully optional condition

To make a given filter's condition fully optional you would use the `'field'` flag. This would mean that these filters are combined with the rest using an `OR`.

This is encoded in the query URL using a tilde before the field name:

```
&f=foo['bar']&f=~marx['groucho']&f=~marx['harpo']*
```

This would result in a query like: `foo:bar) OR (marx:groucho) OR (marx:harpo`

'none' - Logical Conjunction, default (AND)

By default, individual filters (&f=) are applied to the query using logical conjunction (AND). This means that the documents returned must satisfy both filter conditions. This is the default connective.

Alternative way to add Filters

Another way to add Filters to a Query is using dynamic attributes on the Query tag. Any parameters other than the ones detailed above will be treated as field names, and their corresponding value as a filter expression.

For example, this Query tag would create a query from the URL query string and additionally add two hidden filters 1) the source field should contain the value FT and 2) the `bestbets` field should be true:

```
<search:query var="query" parameters="*" results-per-page="5" source="FT"
bestbets="true" />
```

To add exclude filters prefix the filter name with not-. This would add a filter that excludes all documents where the source field contains the value FT:

```
<search:query var="query" parameters="*" not-source="FT" />
```

Tag body

Queries may include the *`query:filter`* tag and the *`query:custom`* tag in the body of a `search:query` tag to add invariant query filters or custom engine parameters, respectively.

```
<search:query var="query" parameters="*" maxResults="5"> <query:filter
field="myField" value="myValue"></query:filter> <query:custom name="facet.range"
value="year"></query:custom> <query:custom name="facet.mincount" value="1">
</query:custom> </search:query>
```

Search Responses

The Response tag (`<search:response>`) sends a Query to a given Platform and stores the Response in a JSP variable.

Usage

This example shows how to submit a query using a given Platform and retrieve the response:

```
<search:response var="myResponse" platform="platform" query="query">
</search:response>
```

The resulting response object can be accessed using an angular expression in the HTML page on which it is located. For example, `{{myResponse}}`.

Configuration parameters

`var (java.lang.String)`
Variable name to reference the Response from the Platform.
Default: `response`

`platform (twigkit.platform.Platform)`
The Platform instance to use.

`query (twigkit.model.Query)`
An instance of an Appkit Query to send to the Platform.

`errorPage (java.lang.String )`
Error page to redirect to when Platform throws an exception.
Default: `/WEB-INF/tags/widgets/error/error.jsp`

`on (java.lang.String)`
Event that is broadcasted for when the response should reload. Default: reloadSearch. (optional)

Tag body

The body of the Response tag can be used to trigger a widget:spinner as the app waits for the response to return from the platform.

```
<search:response var="response" platform="platform" query="query" error-
page="/error"> <widget:spinner></widget:spinner> </search:response>
```

Modules

Appkit's suite of search components is modular, which means you are able to get a working solution into place quickly using tried-and-tested components.

Activity Tracking

Appkit can capture any and all interaction users have with an Appkit powered application. This includes what queries were typed, filters applied, sort order used, and links opened.

The information is stored in a highly scalable embedded search index which can be used to:

- Data mine or analyze information through a regular Appkit user interface
- Empower and aid users in content discovery by grouping together similar queries using statistical methods and suggesting alternatives to users seeking similar content
- Perform auditing and compliance functions in secure applications

Read more

Collaboration

The social and collaboration tools provided by the Social module extend the search application with the ability to store user-generated information on individual documents and document collections, and to share this information among groups of authenticated users.

Features include:

- **Saved searches.** Users can save important keyword search queries, apply filters, and retrieve the information later.
- **Saved results.** Users can bookmark results, add them to a basket, and search those saved items.
- **Rating.** Users can choose to rate or "like" results, giving a social signal to their peers. Rating analytics can be used to influence ranking.
- **Tagging.** Users can tag content with keywords, which content owners can use to improve data quality.
- **Annotations.** Users can annotate/comment on results and edit or delete their own comments. Users cannot edit or delete the comments of others.
- **Topics.** Users can create topics and any captured metadata. For example, saved searches can be associated with a topic.

Read more

Geospatial

Esri Web Maps

The Esri Web Map tag is used within the Geo Map tag to present content from an ArcGIS web map hosted with Esri or on premises.

Maps

The Map tag displays results on a geographical map, abstracting the underlying map provider.

Media

The Media module supports on-the-fly *image transcoding* to mobile formats and resizing including cropping and square cropping. It further supports capturing and transcoding particular pages from multi-page TIFF files and features built in caching. The Media module also supports efficient *exporting search results* to CSV and Excel.

Read more

Export Services

The Excel export service is constrained by the fact that the Excel document needs to be created in memory and therefore recommended for relatively small result sets. For larger exports, we recommend using the CSV export service.

Image Formatting and Transcoding

The Image tag displays images from a given field in a result. It will automatically resize, and cache the image for performance.

Security

Read more

Authentication

OAuth

Authentication against OAuth 2.0 services is possible using the individual authentication provider modules which simplify the configuration of OAuth parameters and deal with the differences that exist between common OAuth endpoints.

A general OAuth provider can be used by including the `twigkit.security.provider.oauth` module. This requires a custom `spring-security.xml` file, but it is recommended to use one of the prepackaged modules we provide for these services.

Read more

SAML

We have a method of configuring SAML authentication with a simplified process. SAML is still a complex standard and so a lot of the relevant information from the previous SAML instructions is still included here.

Appkit provides an extension for SAML based authentication. SAML is a recognized standard for authentication with many providers and implementations available.

Read more

SAML Troubleshooting

The SAML authentication standard is a broad XML specification covering a range of functionality and 'binding' protocols. As such, it is not uncommon to encounter problems integrating applications (Service Provider's) with SAML Identity Providers. Numerous technologies can form the SSO back-end (Identity Provider) and there is no 'one-size-fits-all' approach.

Authorization

Role-based Access

In many cases, you might want to restrict access to your application, or parts of your application, to specific groups of users. To enable this option, you must add the file `conf/security/access.conf` to your application's configuration tree.

Configuration

User Web Service

Although it is disabled by default, there is a web service that returns all of the details for the currently logged in user.

Every operation and capability of Appkit can be invoked via REST-ful web service endpoints supporting both XML and JSON responses for asynchronous client-side operations, and integrating with external systems. For example, users can perform secure searches to any platform supported by Appkit.

Visualizations

The *chart tag library* is a collection of JSP tags for creating charts to visualize both search results (document result sets) and facets.

Read more

Adding Credits

To display credits on the chart in the bottom right hand corner, use the Chart Credit tag with the `text` and `href` attributes.

Chart Legend

A legend is displayed when the chart has multiple series. With the Chart Legend tag, you can customize the legend with a title, border and location on the chart along with other things. This tag must be nested in the *Chart Display tag*.

Chart Tooltips

Tooltips display information when you hover over a point or area of the chart. Tooltips can be set across the whole chart or specifically to a series using the Chart Tooltip tag.

Displaying a Chart

The Chart Display tag displays the facet provided in a chart.

Exporting a Chart

To give users the option to export a chart, include the Chart Export tag within the Chart Display tag. Adding the tag will enable exporting without adding any attributes.

Faceting a Data Series

To render information in facets, use the Series Facet tag, which must be nested within a *Chart Display tag*.

Histogram

The histogram visualization displays the filters of a facet as a left-to-right series of bars.

Search Result Data Series

To render information from fields of a result list into a data series, use the Series Result List tag and use the label-field and value-field attributes.

Tag Cloud

Visualize the top filters of a facet as a tag cloud.

Workflow and Data Processing

A workflow processor is an atomic unit of business logic that can modify a query before it is submitted to a search platform, or transform a search response before rendering. This capability is used extensively for doing runtime data cleaning when, for example, it is not feasible to re-index a whole collection to fix issues in the source data.

Read more

Query Processors

Add a Filter to a Query. By creating or passing in a Filter instance, you can override or set all attributes of the filter that will then be added to the Query.

Read more

Response Processors

Facet Processors

Facet response processors include ones to perform these operations:

- Duplicate a facet.
- Create facets.
- Fill in date filters for a facet.
- Parse and formatting filter values from one Date format to another.
- Create a pseudo date facet.
- Set or changing Parameters for a facet.
- Sort facets based on various properties and parameters.
- Process facet filter values formatted as paths to represent them as hierarchical trees.
- Remove facets from a Response.
- Create pseudo facets based on values in a given field or fields.
- Process hierarchical facets so that only one level is rendered at a time.
- Define a pseudo facet from a list of facets in the response.

Field Processors

Field response processors include ones to perform these operations:

- Capitalise the display value of the given field names.
- Add Highlighting to Fields.
- Tag a document with classifications based on field values.
- Format a date object, replacing the original date value with another.
- Parse dates out of field values.
- Set the value of a field that is missing a value, based on the value of a different field.
- Parse the String value of the specified fields into an Object representation.
- Create a multivalued field from a single field value by using a separator.
- Extract the hostname from URLs and place it in a field named 'site'.
- Process fully qualified URLs in field values and markup, and add anchor tags for active links in the display values.
- Statically add metadata to documents that match a given regular expression.

- Replace field values (actual, display, or both) that are HTML or URL encoded with decoded values.
- Replace field values (actual, display, or both) that match a given regular expression with a different value.
- Make Twitter users and hashtags clickable in the display value.
- Duplicate a field, creating two separate instances.
- Create a new field by joining multiple existing fields using a pattern expression.
- Localize the values of a field using a specified bundle.

Filter Processors

Filter response processors include ones to perform these operations:

- Capitalize the display value of the given filter.
- Parse dates out of facet filter values.
- Sort facet filters by count.
- Sort facet filters by actual value.
- Replace display values in FacetFilters for specified or all Facets.
- Retrieve other filter values for the same facet, despite there being one or more filters for that facet already applied.
- Remove facet filters that match a regular expression pattern from a facet.
- Replace filter values (actual, display, or both) that match a given regular expression with a different value.
- (Since 4.2.0) Dynamically format the display value of a date range facet according to the size of the range.

General Response Processors

General response processors include ones to perform these operations:

- Perform a left join on fields.
- Log response metadata and attributes to log4j.
- Group results by a given Field value.

Activity Tracking

Appkit can capture any and all interaction users have with an Appkit powered application. This includes what queries were typed, filters applied, sort order used, and links opened.

The information is stored in a highly scalable embedded search index which can be used to:

- Data mine or analyze information through a regular Appkit user interface
- Empower and aid users in content discovery by grouping together similar queries using statistical methods and suggesting alternatives to users seeking similar content
- Perform auditing and compliance functions in secure applications

Activity processing

The activity tracking framework supports these endpoints for processing activity events:

- *Lucidworks Fusion Signals*
- *Log file writer*

Query Tracking

To track user-generated queries on your page, add a `<track:query>` tag with a reference to the query object that captures the user's input:

```
<track:query query="query" path="/search" page-title="Search Page"></track:query>
```

To see all related attributes, see the *track:query* tag documentation.

Click Tracking

To track click events, wrap a `<track:clicks>` tag around any result list to be tracked:

```
<track:clicks query="query" type="People Results"> <search:result-list
response="response"> <search:result> ... </search:result> </search:result-list>
</track:clicks>
```

To see all related attributes, see the *track:clicks* tag documentation.

Alternatively, to track a static URL, wrap the address in your href as shown. Optionally, you can use the `_type` parameter to provide a click type:

```
/twigkit/secure/services/url?url=http://www.google.com&_type=Other_Sites"
```

Visualizations

The *chart tag library* is a collection of JSP tags for creating charts to visualize both search results (document result sets) and facets.

Usage

You can visualize your data using the charts markup language. As an example, let us say you want to plot distinct segments of a search response facet as a chart, with each segment shown in proportion to size of that segment. For this you would add the *Chart Display* tag to your page, passing in a reference to the `twigkit.model.Response` containing the facet you want to plot and specify the facet name and type of chart you want to render. For example:

```
<chart:display response="response" facet-name="current_products" title="Current Products" type="pie"></chart:display>
```

In some cases, you might want to specify more options for your chart, or provide multiple data series to plot, for example. You do that by opening the `chart:display` tag and providing one or more chart options or tags in the markup body:

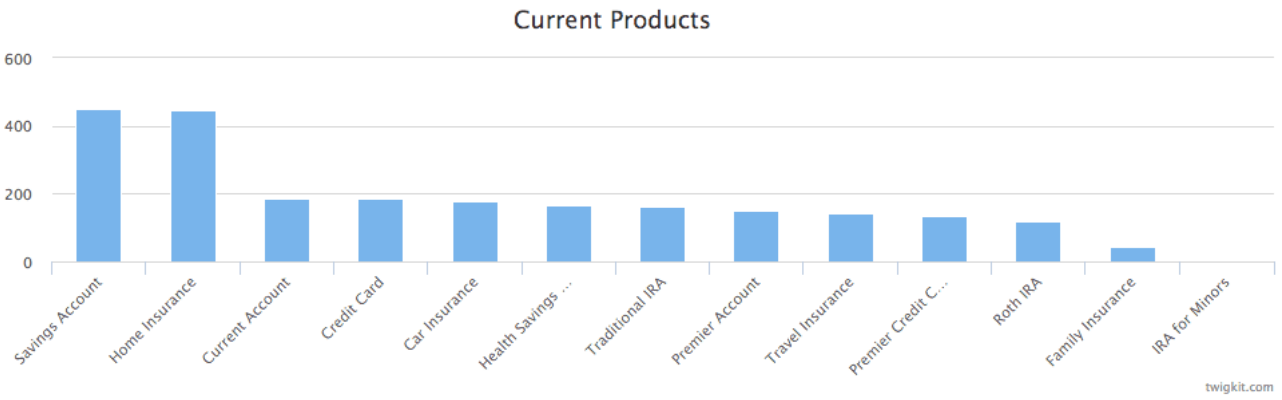
```
<chart:display response="response" title="Current Products"> <series:facet name="current_products" type="column" title="Current Products"></series:facet> <chart:tooltip header-format="<b>{point.x}</b><br/>" point-format="{series.name}: <b>{point.y}</b><br/>"></chart:tooltip> </chart:display>
```

Adding Credits

To display credits on the chart in the bottom right hand corner, use the Chart Credit tag with the `text` and `href` attributes.

This tag must be nested in the *Chart Display tag*.

```
<chart:display response="response" title="Current Products" type="column" facet-name="current_products"> <chart:credit href="http://twigkit.com" text="twigkit.com"></chart:credit> </chart:display>
```



To see all related attributes, see the *chart:credit* tag documentation. link:/fusion/5.4/reference/tags/lightning.directive

Chart Legend

A legend is displayed when the chart has multiple series. With the Chart Legend tag, you can customize the legend with a title, border and location on the chart along with other things. This tag must be nested in the *Chart Display tag*.

```
<chart:display response="response" title="Current Products vs Recommended Products" stacking="normal"> <series:facet name="current_products" type="column" title="Current Products"></series:facet> <series:facet name="recommended_products" type="column" title="Recommended Products"></series:facet> <chart:legend title="Legend Title" align="right" vertical-align="top" layout="vertical" y="100" border-radius="5" border-width="1"></chart:legend> </chart:display>
```

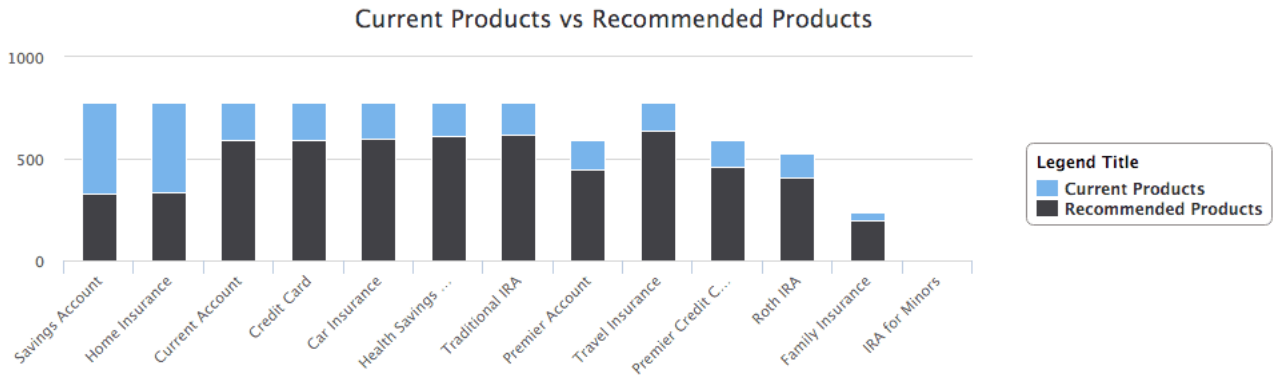


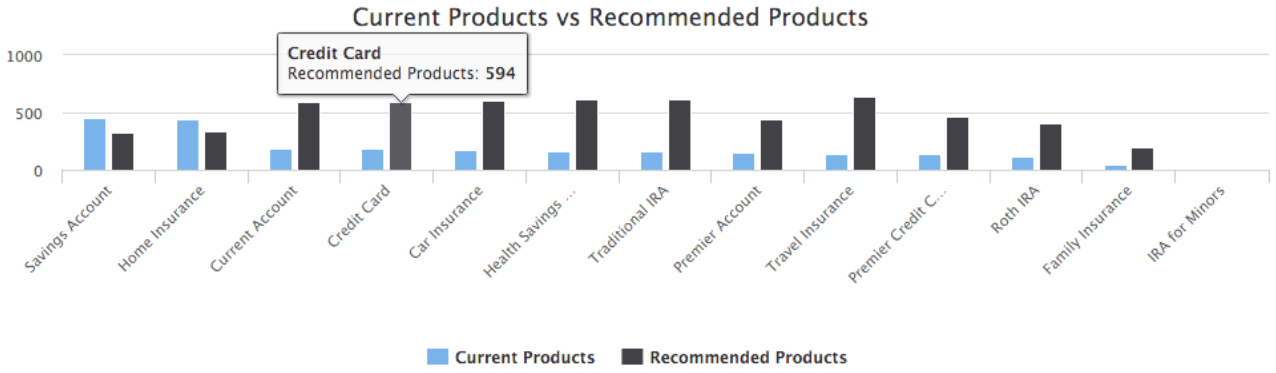
Chart Tooltips

Tooltips

Tooltips display information when you hover over a point or area of the chart. Tooltips can be set across the whole chart or specifically to a series using the Chart Tooltip tag.

This tag must be nested in the *Chart Display tag*, or in either of the series tags (*Series Facet tag* and *Series Result List tag*) if the tooltip is specific to a series.

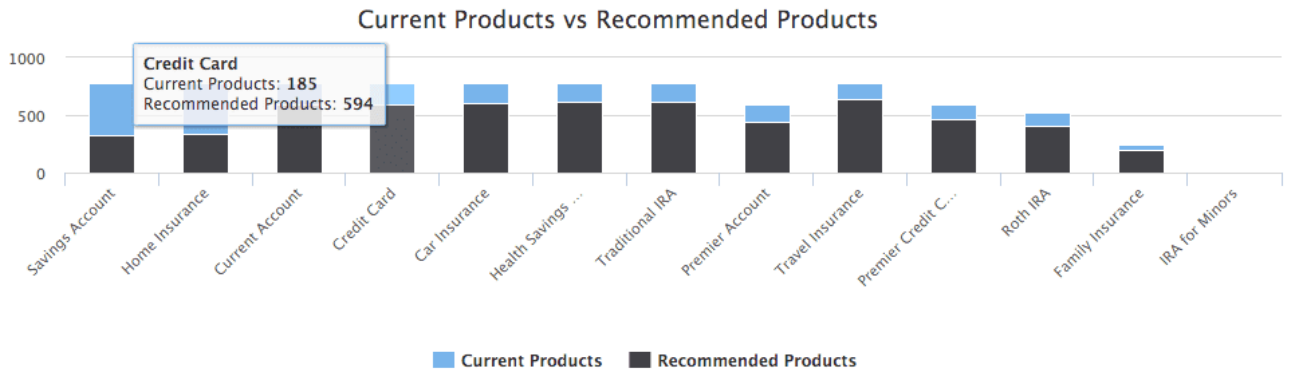
```
<chart:display response="response" title="Current Products vs Recommended
Products"> <series:facet name="current_products" type="column" title="Current
Products"></series:facet> <series:facet name="recommended_products" type="column"
title="Recommended Products"></series:facet> <chart:tooltip header-format="<b>
{point.x}</b><br/>" point-format="{series.name}: <b>{point.y}</b><br/>">
</chart:tooltip> </chart:display>
```



Shared Tooltip

If you are rendering more then one series you can set the tooltip to be shared, this means it will display information about all the series for the point you have hovered over.

```
<chart:display response="response" title="Current Products vs Recommended Products"
stacking="normal"> <series:facet name="current_products" type="column"
title="Current Products"></series:facet> <series:facet name="recommended_products"
type="column" title="Recommended Products"></series:facet> <chart:tooltip
headerFormat="<b>{point.x}</b><br/>" point-format="{series.name}: <b>{point.y}</b>
<br/>" shared="true"></chart:tooltip> </chart:display>
```



To see all related attributes, see the *chart:tooltip* tag documentation.

Displaying a Chart

The Chart Display tag displays the facet provided in a chart.

The body of the Chart Display tag can contain a *Series Facet* tag for rendering multiple series using facets and also a *Series Result List* tag for rendering multiple series using results.

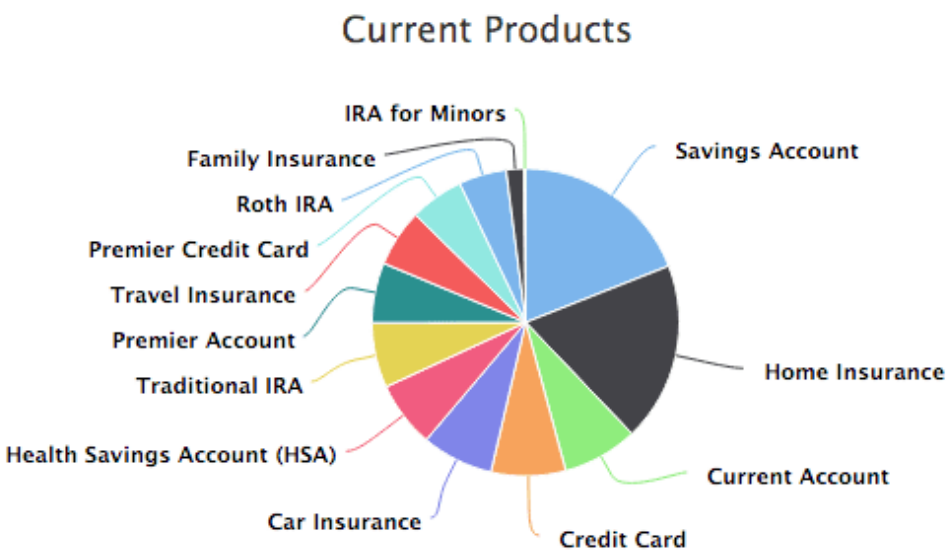
Along with this, there are other tags to customize the chart. These include the *Chart Tooltip* tag for modifying the tooltip, *Chart Legend* tag for modifying the legend, *Chart Credit* tag for adding a credit to the chart, and the *Chart Export* tag for exporting the chart.

Usage

Displaying a facet

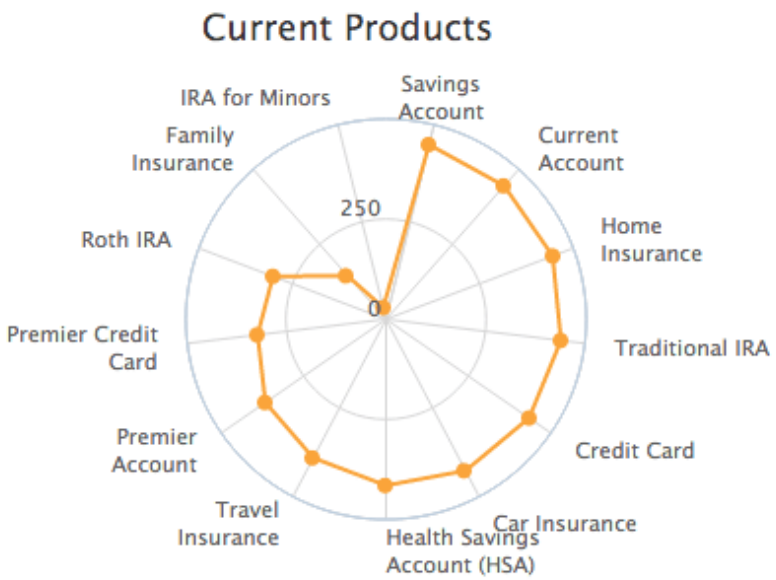
To visualize information in facets using frequency to build the graph, use the facetName attribute:

```
<chart:display response="response" facet-name="current_products" title="Current
Products" type="pie"></chart:display>
```



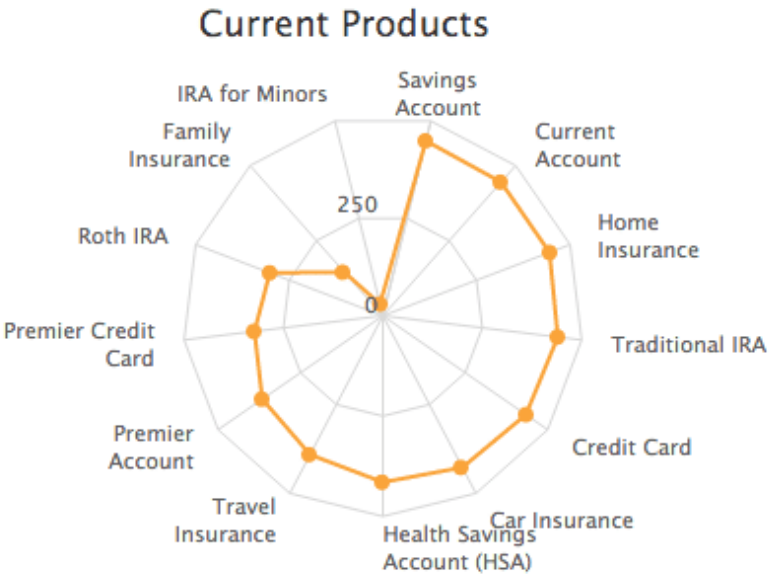
Displaying a polar chart

```
<chart:display response="response" facet-name="current_products" title="Current
Products" type="line" radar-or-polar="polar"></chart:display>
```



Displaying a radar chart

```
<chart:display response="response" facet-name="current_products" title="Current Products" type="line" radar-or-polar="radar"></chart:display>
```



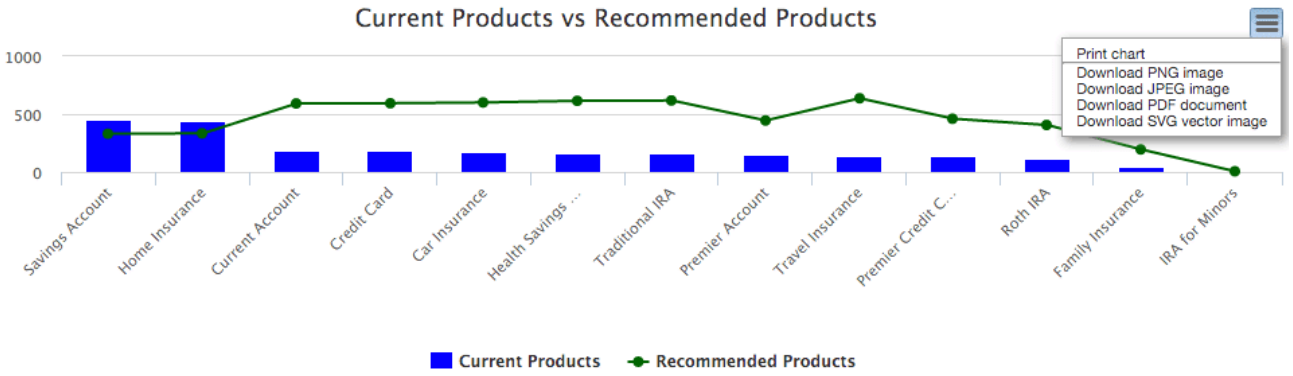
To see all related attributes, see the *chart:credit* tag documentation.

Exporting a Chart

To give users the option to export a chart, include the Chart Export tag within the Chart Display tag. Adding the tag will enable exporting without adding any attributes.

This tag must be nested in the *Chart Display tag*.

```
<chart:display response="response" title="Current Products vs Recommended
Products"> <series:facet name="current_products" type="column" title="Current
Products" color="blue"></series:facet> <series:facet name="recommended_products"
type="line" title="Recommended Products" color="#006600"></series:facet>
<chart:export></chart:export> </chart:display>
```



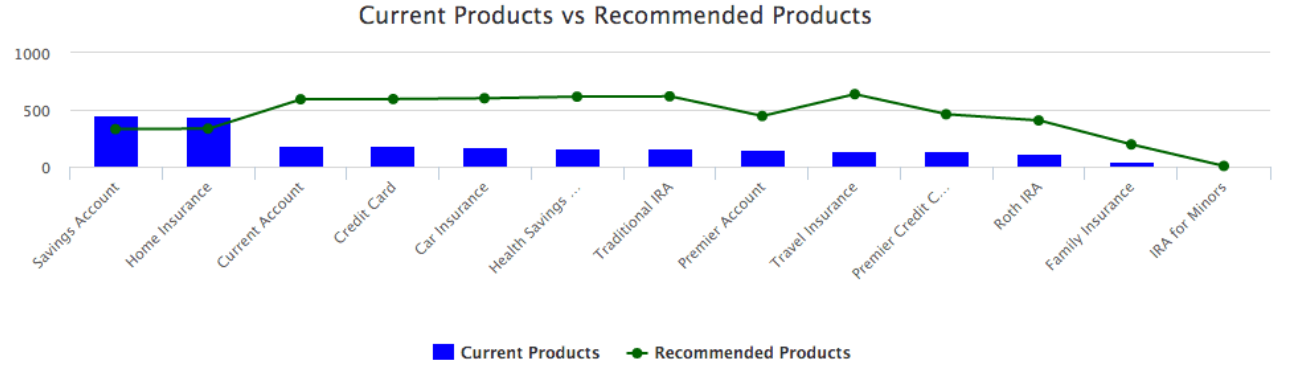
To see all related attributes, see the *chart:export* tag documentation.

Faceting a Data Series

To render information in facets, use the Series Facet tag, which must be nested within a *Chart Display tag*.

Usage

```
<chart:display response="response" title="Current Products vs Recommended
Products"> <series:facet name="current_products" type="column" title="Current
Products" color="blue"</series:facet> <series:facet name="recommended_products"
type="line" title="Recommended Products" color="#006600"</series:facet>
</chart:display>
```



To see all related attributes, see the *series:facet* tag documentation.

Histogram

The histogram visualization displays the filters of a facet as a left-to-right series of bars.

Usage

Simply pass a facet name into the histogram visualization to see that facet presented as a histogram chart.

```
<chart:histogram facet-name="facet" show="20"></chart:histogram>
```

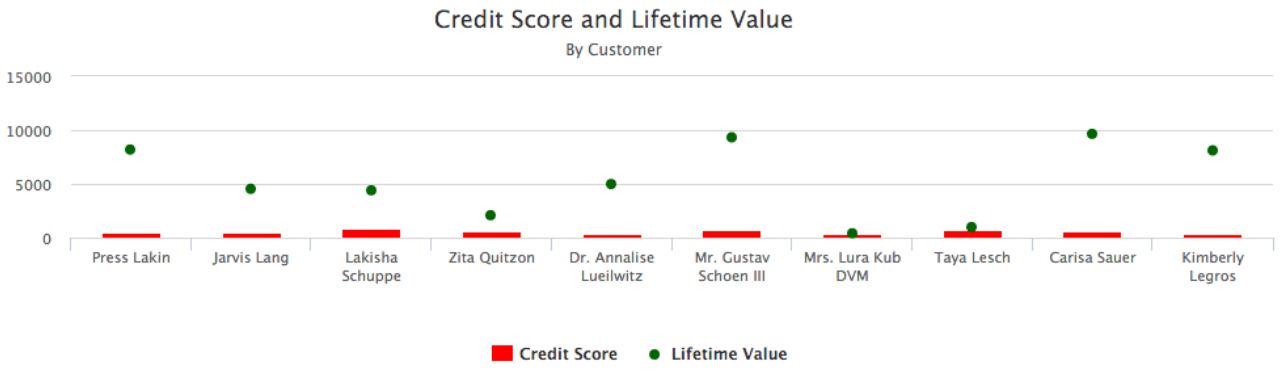
To see all related attributes, see the *chart:histogram* tag documentation.

Search Result Data Series

To render information from fields of a result list into a data series, use the Series Result List tag and use the label-field and value-field attributes.

This tag must be nested in the *Chart Display tag*.

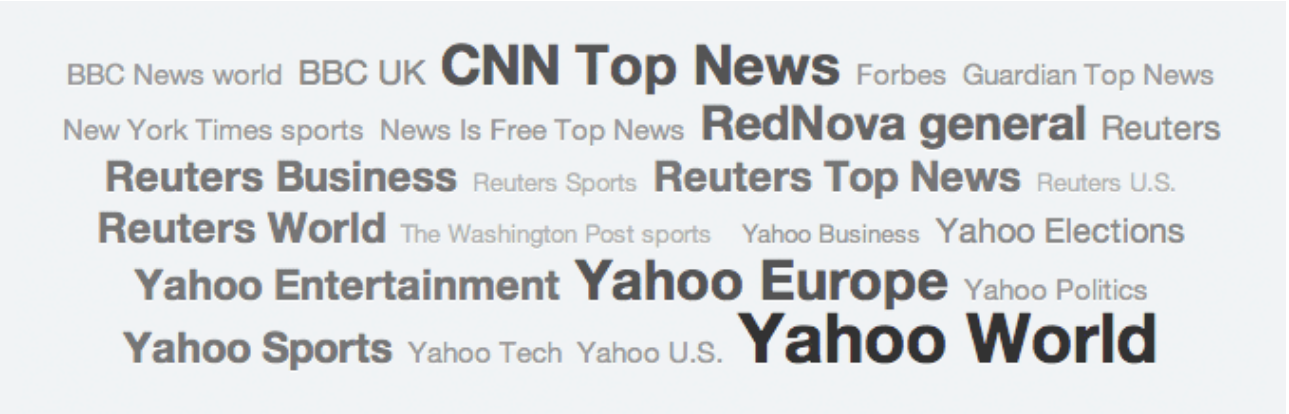
```
<chart:display response="response" title="Credit Score and Lifetime Value"
subtitle="By Customer"> <series:result-list label-field="name" value-
field="credit_score" type="column" title="Credit Score" color="red">
</series:result-list> <series:result-list label-field="name" value-
field="lifetime_value" type="scatter" title="Lifetime Value" color="#006600">
</series:result-list> </chart:display>
```



To see all related attributes, see the *series:result-list* tag documentation.

Tag Cloud

Visualize the top filters of a facet as a tag cloud.



Usage

To add a tag cloud, specify the response, which facet to display, and (optionally) how many tags to display.

```
<chart:tagcloud response="response" facet-name="facet"></chart:tagcloud>
```

To see all related attributes, see the *chart:tagcloud* tag documentation.

Security

Authentication
The Security Module provides authentication and authorization capabilities to the search application. It supports multiple authentication providers and will pass authentication credentials on to search engines capable of handling authorization (for example, the FAST Security Access Module), providing personalization and role based filtering. The Security Module can be extended and customized via a pluggable API.

The Security module supports a number of authentication providers, including:

- **Single sign-on** via Windows Integrated Authentication
- **Container-managed** authentication
- **Form-based** authentication
- **Cookie-based** authentication
- **Pass-through** authentication
- **Custom authentication providers** via API

Role-based Access
In many cases, you might want to restrict access to your application, or parts of your application, to specific groups of users. To enable this option, you must add the file `conf/security/access.conf` to your application's configuration tree.

User Web Service
Although it is disabled by default, there is a web service that returns all of the details for the currently logged in user.

Every operation and capability of Appkit can be invoked via REST-ful web service endpoints supporting both XML and JSON responses for asynchronous client-side operations, and integrating with external systems. For example, users can perform secure searches to any platform supported by Appkit.

Authentication

The Security Module provides authentication and authorization capabilities to the search application. It supports multiple authentication providers and will pass authentication credentials on to search engines capable of handling authorization (for example, the FAST Security Access Module), providing personalization and role based filtering. The Security Module can be extended and customized via a pluggable API.

The Security module supports a number of authentication providers, including:

- **Single sign-on** via Windows Integrated Authentication
- **Container-managed** authentication
- **Form-based** authentication
- **Cookie-based** authentication
- **Pass-through** authentication
- **Custom authentication providers** via API

Authentication methods

OAuth

Authentication against OAuth 2.0 services is possible using the individual authentication provider modules which simplify the configuration of OAuth parameters and deal with the differences that exist between common OAuth endpoints.

A general OAuth provider can be used by including the `twigkit.security.provider.oauth` module. This requires a custom `spring-security.xml` file, but it is recommended to use one of the prepackaged modules we provide for these services.

Read more about OAuth authentication.

SAML

We have a method of configuring SAML authentication with a simplified process. SAML is still a complex standard and so a lot of the relevant information from the previous SAML instructions is still included here.

Appkit provides an extension for SAML based authentication. SAML is a recognized standard for authentication with many providers and implementations available.

Read more about SAML authentication.

Available security providers

Appkit supports a range of security providers for authentication and authorization. Typically one of these providers is used in an application:

Active Directory

While integration with Active Directory can be achieved using the LDAP setup, Spring Security provides a tailored security provider which makes configuring security using Active Directory easier.

Set Up Active Directory Authentication

Fusion Sessions API

The Fusion security provider for Appkit allows authentication against Fusion using the Fusion Sessions API. With this provider configured, an application deployed within a Fusion defined 'realm' such as `native` can securely authenticate a user against Fusion.

Set Up Lucidworks Fusion Authentication

HTTP Headers

Appkit supports authentication via generic HTTP headers passed through from a component behind a secure firewall sitting in front of the Appkit application. If there is such an authentication component in the environment where you are using Appkit, you can configure the HTTP headers that the application will use to pick up the user's details.

Set Up HTTP Headers Authentication

Kerberos

The Kerberos security provider for Appkit allows authentication using Kerberos. With this provider configured, an application deployed within a Kerberos 'realm' can securely authenticate a user against a Kerberos 'Key Distribution Center'. This module is primarily

intended to simplify the process of authenticating a user silently in a Windows environment by leveraging the built in support for integrated Windows authentication in Internet Explorer. However it should be possible to use this module to authenticate the user in any Kerberos compliant environment.

Set Up Kerberos Authentication

LDAP

Appkit allows for authentication against a server using LDAP (Lightweight Directory Access Protocol). It assumes that your Appkit application is managed as a Maven *(https://maven.apache.org/)* project with all project dependencies described in a Maven POM file *(https://maven.apache.org/pom.html)*.

Set Up LDAP Authentication

Siteminder

Appkit can integrate with a SiteMinder single-sign-on authentication provider. SiteMinder is typically installed in a web server placed in front of the application server that runs Appkit.

Set Up SiteMinder Authentication

Spring Security

The Spring Security authentication provider for Appkit interfaces with both Spring Security *(http://projects.spring.io/spring-security/)* 3 and 4. Projects that adopt Appkit version 3 must switch to using Spring Security 4.

Authorized Spring principals become fully-qualified Appkit user profiles, along with any roles or groups that the Spring module provides. Spring Security integrates with a number of authentication protocols, including LDAP, Active Directory, Basic Authentication, Kerberos, JAAS, and others. For a full list, see here. *(http://docs.spring.io/spring-security/site/docs/3.1.1.RELEASE/reference/introduction.html)*

Set Up Spring Authentication

Tomcat Container

Apache Tomcat *(https://hub.docker.com/_/tomcat)* is an open source Java servlet container. Tomcat implements Java enterprise specs, including the Java Servlet, JavaServer Pages JSP, and WebSockets APIs.

Set Up Tomcat Container Authentication

Windows

Appkit can be used with Integrated Windows Authentication *(http://en.wikipedia.org/wiki/Integrated_Windows_Authentication)* for single sign on (SSO) *(http://en.wikipedia.org/wiki/Single_sign-on)* in your search application using Microsoft Internet Information Services *(http://www.iis.net/)*.

Set Up Windows Authentication

Configuration

Primary configuration is done via the `security.conf` configuration file and other files in the `/conf/security/` folder. The Security Module however also supports mechanisms like Spring Security that might require its own configuration files (for example, spring-security.xml) and custom elements in `web.xml`.

Define the security provider in `security.conf` using one of the following options:

```
type: spring_security type: saml type: kerberos type: tomcat7 type: generic type:
siteminder type: passthrough type: oauth
```

Spring security

Spring Security is an industry standard security framework for Java. It is a popular option allowing the security to be configured initially via a simple XML file, and offers a lot of flexibility due to its integration with many different technologies, supporting a range

of security standards. The spring documentation *(http://projects.spring.io/spring-security/)* contains thorough guides to the use of Spring Security.

By default Appkit applications are configured to use Spring Security with a simple hard-coded set of credentials in the provided `spring-security.xml` file, but you can see some other common configurations *defined here*. Further documentation on configuring Spring Security can be found here *(http://docs.spring.io/spring-security/site/docs/3.0.x/reference/ns-config.html)*.

Appkit can be configured to use Spring Security for both authentication and role extraction.

Container managed security

In this case the application server is responsible for authentication such as:

- Internet Information Server and Integrated Windows Authentication
- *Tomcat 7 Container Managed*

The application server can be configured to pick up the user's credentials when the request is forwarded from a 'front-end' like IIS or Apache. In cases where authentication is done by the container it is still possible to do role extraction using Active Directory or LDAP for the user provider (see below).

Generic security

The generic security option uses getRemoteUser() or the REMOTE_USER variable provided through the servlet API. This is for authentication only, but it can be combined with other methods for picking up the roles assigned to a user and submitting them to the search platform (see Active Directory and LDAP role extraction below).

Custom Providers

You can add your own SecurityProvider which would be responsible for providing the user identity to Appkit. To do this, implement the `twigkit.security.SecurityProvider` interface and bind the class using Guice in your custom module.

Authorization, Role Extraction, and other features

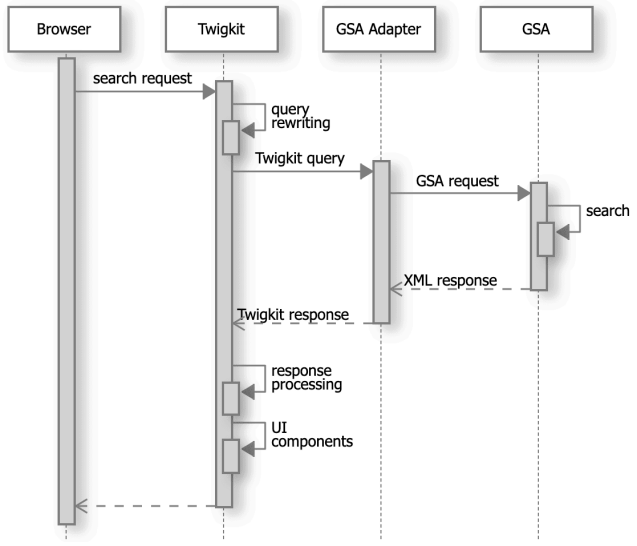
Active Directory and LDAP Role Extraction

After the user's identity has been established during the authentication stage, it is possible to carry out further authorization tasks such as role extraction or adding other metadata about the current user.

User and Domain Mapping

Appkit Security Module comes with pre-built filters to handle mapping between user names and domains. It is also possible to provide your own logic during this phase to deal with specific scenarios for determining a user's credentials and details before rendering the page or passing them to the search platform.

Security lifecycle



1. Authentication
When the Security Module is installed, every request made to Appkit is filtered through the configured authentication provider.

Credentials are added to the request via a ThreadLocal variable. Authentication providers can be container managed, single sign on or configured for form based authentication in Appkit. You can create your own authentication provider by implementing and binding the twigkit.security.SecurityProvider class.

2. Role Extraction
Next step in the filter chain can handle role/group extraction for the authenticated user. This is frequently done via an Active Directory/LDAP lookup. The role extraction mechanisms will add the roles for the user to the User object.

3. Query
When the Security Module is used queries should be created using the secure version of the Query tag. The secure query tag will provide a reference to the user that is logged in and pass the identity on to the search platform if necessary. The Security Module also provides a number of processors and utilities for adding filters to the query based on business rules or identity/role information.

4. Personalization
The User and Role information is available in the user interface. Using the Security Conditional tag you can customize the view to the user that is logged in.

OAuth

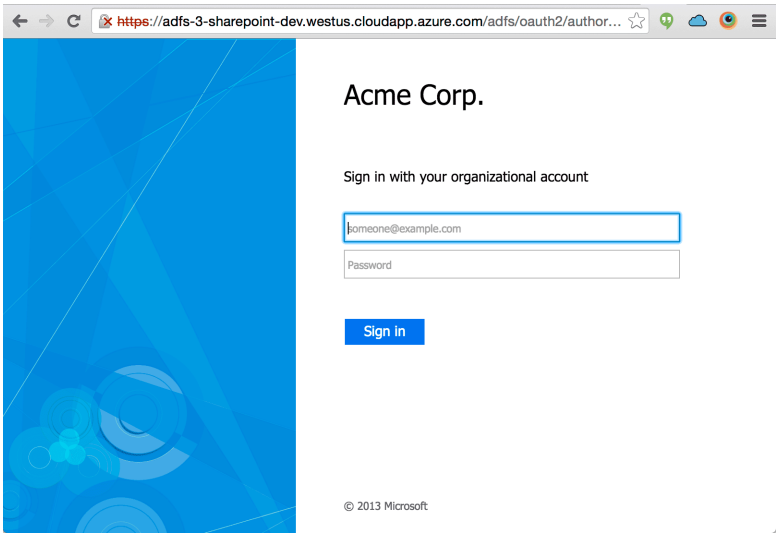
Authentication against OAuth 2.0 services is possible using the individual authentication provider modules which simplify the configuration of OAuth parameters and deal with the differences that exist between common OAuth endpoints.

A general OAuth provider can be used by including the `twigkit.security.provider.oauth` module. This requires a custom `spring-security.xml` file, but it is recommended to use one of the prepackaged modules we provide for these services.

Using OAuth can create a seamless experience where, after the token handshake and approval is complete, the user will always be logged into the application as long as their session is active.

ADFS

The Active Directory Federation Services (ADFS) security provider is available in Appkit.



OAuth based authentication against ADFS requires **ADFS 3** (or newer) which is available from Windows Server 2012 R2 onwards.

Learn how to set up ADFS authentication.

Facebook

Learn how to set up Facebook authentication.

Google

Learn how to set up Google authentication.

Office 365

Learn how to set up Office 365 authentication.

SAML Configuration

We have a method of configuring SAML authentication with a simplified process. SAML is still a complex standard and so a lot of the relevant information from the previous SAML instructions is still included here.

Appkit provides an extension for SAML based authentication. SAML is a recognized standard for authentication with many providers and implementations available.

To learn how to configure SAML based authentication, see the *how-to article*.

Typically SAML provides Single-Sign-On (SSO) between services that can be accessed using the same identity. This might, for example, be services provided by different government departments, or different sites in a company's intranet.

An entity that provides the back-end system determining a user's access and credentials is known as the **identity provider (IdP)**. The individual web resources that a user might want to access are referred to as **service providers (SP)**.

Generally one IdP provides the SSO login form and numerous SPs that can all be used after the user has logged into the IdP's login form. However, federated configurations are also common where either single or multiple login forms can enable authentication against multiple IdPs. In this case, there will often be some 'middleware' that mediates the login requests.

Additional configuration

If additional customizations to the configuration are needed, *download the default spring-security.xml file for SAML*. This adds all the required filters for the SAML extension to function, but it will need customising for the security configuration the application is to authenticate against.

Additional parameters

Some additional parameters can be used in `saml.conf` to change the behavior of the SAML module:

`metadata-display`: set to 'metadataDisplayFilter' and you will be able to download an SP metadata XML file at `http://your-twigkit-host/your-twigkit-app/saml/metadata`. Thid id useful for debugging, but should be removed for production deployments because it exposes information about the authentication protocol.

`saml-keystore-file`: location of the keystore file to use for checking trust and identity of the IdP and SP security certificates

`key-alias`: the alias of a key in the keystore to use as the default SP security key

`key-password`: password for the keystore and the key defined under using 'key-alias' (must match)

`max-authentication-age`: a TTL for the authentication in seconds, useful if the authentication expires at the IdP end prior to the default which is 7200 seconds.

`response-skew`: a time window in seconds to accept initial SAML responses from the IdP. This is useful if the IdP server and application server are on different timezones.
Default: 60 seconds

Configure for a load balancer

When using a load balancer or reverse-proxy in front of an App Studio web app, you must *download a modified spring-security.xml file*. You can then add these additional attributes to `saml.conf` with the respective connection values for your load balancer:

`server-scheme`: The load balancer's communication protocol (for example, http)

`server-name`: The load balancer's host name (for example, search.twigkit.com)

`server-port`: The load balancer's port number

`entity-base-url`: A full URL for the load balancer (for example, http://search.twigkit.com *(http://search.twigkit.com)*)

Troubleshooting SAML

The SAML authentication standard is a broad XML specification covering a range of functionality and 'binding' protocols. As such, it is not uncommon to encounter problems integrating applications (Service Provider's) with SAML Identity Providers. Numerous technologies can form the SSO back-end (Identity Provider) and there is no 'one-size-fits-all' approach.

If you have reviewed the main SAML documentation for integration in Appkit and are encountering problems, here we document tips and tricks picked up from previous installations.

Because Appkit uses the Spring Security SAML *(http://docs.spring.io/spring-security-saml/docs/1.0.x/reference/htmlsingle/)* extension for underlying SAML compliant client behavior, a useful reference is the Spring documentation *(http://docs.spring.io/spring-security-saml/docs/1.0.x/reference/htmlsingle/)*.

Debugging

To enable detailed debugging, add this to your `logback.xml` file:

```
<logger name="org.opensaml" level="TRACE"/> <logger
name="org.springframework.security" level="TRACE"/>
```

You will now see the content of any SAML request and response messages, as well as a lot of extra debugging information. Also, if you suspect a certificate or SSL related error, you can start the JVM/Tomcat with this argument:

```
-Djava.net.debug=ssl
```

IdP rejects an authentication request (AutnRequest) from app

A range of issues could cause this. Most commonly, it is because the request from the application failed validation which is often due to the IdP lacking the correct configuration. You must ensure that the endpoints and application identifiers match in the IdP's configuration for the application. For example if using ADFS, this is the 'Relying Party Trust' configuration. If manually configuring this is proves problematic, you can ensure that the IdP has the correct configuration for the app by generating an SP 'metadata' XML file that can be imported into the IdP.

To enable this you must:

1. Add this to `spring-security.xml` in the filter chain:

   ```
   <filter-chain pattern="/saml/metadata/**" filters="metadataDisplayFilter"/>
   ```

   For example:

   ```
   ...
   <filter-chain pattern="/saml/logout/**" filters="samlLogoutFilter"/>
   <filter-chain pattern="/saml/metadata/**" filters="metadataDisplayFilter"/>
   <filter-chain pattern="/saml/SSO/**" filters="samlWebSSOProcessingFilter"/>
   ...
   ```

2. Add the bean definition for this filter:

```
<!-- The filter is waiting for connections on URL suffixed with filterSuffix and
presents SP metadata there -->
<beans:bean id="metadataDisplayFilter"
class="org.springframework.security.saml.metadata.MetadataDisplayFilter"/>
```

3. Deploy the application and browse to `http://your-app-server/saml/metadata/`.
   This should generate the SP metadata XML file describing the applications configuration which can then be imported to almost any compliant IdP in the administration console/configuration.

After the issue is resolved in production, we recommend that you remove this filter because it exposes information about the authentication mechanism.

IdP rejects authentication or sends back a status code in response message indicating an error

It is worth inspecting the status code in the response message and the logs from the IdP as a next step. This issue is often found if the configuration matches at both the IdP and SP, but there is a 'trust' issue, which can be related to the signing certificates used. We have found some IdPs (ADFS for example) that reject self-signed certificates from the application. It is recommended to use a genuine trusted certificate to avoid this problem.

Illegal Key Size error

You might find you get an error at some stage that is a `java.security.InvalidKeyException` or similar stating 'Illegal Key Size'. This is because a key has been used with a size larger than that which the JVM accepts by default. This is a known Java issue to do with cryptography restrictions shipped with a standard JVM. The solution is to install the 'unrestricted JCE policy files' in the JVM on which the application is running. You can obtain this from Oracle and it is usually installed in the JRE directory in `lib/security`. Make sure you install the files for the correct version of Java.

Fusion rejects a SAML assertion

There are a few reasons why Fusion might reject a SAML assertion. If you're having issues:

- Check that **IdP Issuer** matches `<saml:Issuer>` in the SAML payload.
- Check that **App Issuer** matches `<saml:Audience>` in the SAML payload.
- When using encrypted certificates, you must provide the X.509 key info in the SAML payload, which must match the **Certificate Fingerprint** in Fusion.

Refer to *Configure Fusion for SAML* for more information.

Role-based Access

Since 4.2.0

In many cases, you might want to restrict access to your application, or parts of your application, to specific groups of users. To enable this option, you must add the file `conf/security/access.conf` to your application's configuration tree.

Use the following configuration parameters:

```
allow: role-with-access-1,role-with-access-2 deny: role-without-access-1,role-without-access-2 pattern: regex-of-uris-to-intercept
```

These configuration parameters are:

- `allow`: A comma-separated list of roles that should be granted access. When not specified or set to a wildcard ('*'), Appkit defaults to allowing access to all roles.
- `deny`: A comma-separated list of roles that should be denied access.
- `pattern`: A regular expression defining the pattern of URIs that should be intercepted for checking access. This defaults to .* (all paths are checked).

Example

Let us assume you have a user directory containing two user, `user` and `admin`, where only the latter has an `ADMIN` role. For example, you can define this using a simple `spring-security.xml` configuration with a static list of users like so:

```
<authentication-manager> <authentication-provider> <user-service> <user name="user" password="user" authorities="USER"/> <user name="admin" password="admin" authorities="USER,ADMIN"/> </user-service> </authentication-provider> </authentication-manager>
```

To configure role-based access, you add `conf/security/access.conf` to the application with this configuration:

```
allow: ADMIN pattern: (/)|(/twigkit/api/.*)
```

This says that for all requests to `/` (root page) and the API service we apply role-based authorization rules. The former is strictly not necessary, just leads to better UX for users denied access.

To validate this setup, first log in as `user` and get denied access, as shown in the application logs:

```
TRACE t.s.SecurityFilterExecutionModule - Filtering with [twigkit.security.filter.RoleBasedAuthorizationFilter] TRACE t.s.f.InterceptAuthorizationFilter - Filtering request to / - comparing against access pattern (/)|(/twigkit/api/.*) TRACE t.s.f.RoleBasedAuthorizationFilter - User 'user' is DENIED access to protected resource ERROR t.s.SecurityFilterExecutionModule - Authorization chain failed - returning 403
```

Subsequently, log in as `admin` and get access to the app, as shown in the logs:

```
TRACE t.s.SecurityFilterExecutionModule - Filtering with
[twigkit.security.filter.RoleBasedAuthorizationFilter] TRACE
t.s.f.InterceptAuthorizationFilter - Filtering request to / - comparing against
access pattern (/)|(/twigkit/api/.*) TRACE t.s.f.RoleBasedAuthorizationFilter -
User 'admin' is GRANTED access to protected resource
```

User Web Service

Although it is disabled by default, there is a web service that returns all of the details for the currently logged in user.

Every operation and capability of Appkit can be invoked via REST-ful web service endpoints supporting both XML and JSON responses for asynchronous client-side operations, and integrating with external systems. For example, users can perform secure searches to any platform supported by Appkit.

Invoke the User web service by adding this parameter to `conf/security.conf`:

```
webservice-enabled: true
```

Access the User web service via GET request at `/twigkit/api/user` for example:

```
http://my-twigkit-host/my-twigkit/twigkit/api/user
```

See also *How to Import Security Certificates*.

Geospatial Module

The articles in this section describe the Geospatial module:

- *ESRI Web Maps*
  The Esri Web Map tag is used within the Geo Map tag to present content from an ArcGIS web map hosted with Esri or on premises.
- *Maps*
  The Map tag displays results on a geographical map, abstracting the underlying map provider.

Esri Web Maps

The Esri Web Map tag is used within the Geo Map tag to present content from an ArcGIS web map hosted with Esri or on premises.

Usage

If the web map is hosted with Esri, you can add the web map without a portal URL by specifying the web map ID. (For example, http://www.arcgis.com/home/webmap/viewer.html?webmap=de87749b005346d59e1cd34275154c96 *(http://www.arcgis.com/home/webmap/viewer.html?webmap=de87749b005346d59e1cd34275154c96)*)

```
<geo:map provider="esri" response="response" address-field="customer_address"
label-field="name" height="700px"> <esri:webmap
id="de87749b005346d59e1cd34275154c96"></esri:webmap> </geo:map>
```

If the web map is hosted on your own servers, simply specify the web map ID and the URL of the server; for example:

```
<geo:map provider="esri" response="response" address-field="customer_address"
label-field="name" height="700px"> <esri:webmap
id="de87749b005346d59e1cd34275154c96"
url="http://yourportalurl.com/sharing/rest/content/items"></esri:webmap> </geo:map>
```

To see all related attributes, see the *esri:webmap* tag documentation.

Maps

The Map tag displays results on a geographical map, abstracting the underlying map provider.

Usage

If the results contain latitude and longitude values as fields you can render the results using the Google Map API with this configuration:

```
<geo:map provider="google" response="response" latitude-field="MT.lat" longitude-
field="MT.lng"></geo:map>
```

If you do not have coordinates in the results, then the Map tag can do a geocoding of addresses and render these on a map:

```
<geo:map provider="bing" response="response" address-field="fulladdress">
</geo:map>
```

To see all related attributes, see the *geo:map* tag documentation.

Media Module

The Media module supports on-the-fly *image transcoding* to mobile formats and resizing including cropping and square cropping. It further supports capturing and transcoding particular pages from multi-page TIFF files and features built in caching. The Media module also supports efficient *exporting search results* to CSV and Excel.

Adding the media module

Add the media module dependency to your `pom.xml` file:

```
<dependency> <groupId>twigkit</groupId> <artifactId>twigkit.media</artifactId>
<version>${project.parent.version}</version> </dependency>
```

Export Services

Exporting results to Excel

The Excel export service is constrained by the fact that the Excel document needs to be created in memory and therefore recommended for relatively small result sets. For larger exports, we recommend using the CSV export service.

Use the `media:excel` tag to generate a link to export results to Excel.

```
<media:excel conf="platforms.gsa.foo" query="query" fetch="10000"
fields="title,foo" header="myTitle,myFoo">Export</media:excel>
```

Exporting results to CSV

The CSV export service is used for writing search results to a file in comma-delimited format (or other delimited formats). This service requires the platform configuration to use and any combination of query parameters you would normally pass to an Appkit search page.

The CSV export service is highly scalable and will retrieve entire indexes (if deep offset is supported by the search engine) and spool to files in a scalable manner.

You can generate an export link using the `media:csv` tag:

```
<media:csv conf="platforms.gsa" query="query" delimiter="|" fetch="100000" batch-
size="100" fields="foo,bar,zip">Export to CSV</media:csv>
```

Offline export

When exporting a large number of documents, it can be preferable to run the export process in the background, to avoid browser time-out. We cater for this possibility by running *offline* exports in a separate thread and storing the CSV output in a file in the local file system of the application server (which might be a shared mount when running multiple nodes). To control how export files are generated and stored, you have these configuration options in `services/media/export/offline` in the configuration hierarchy:

- `compress`: Whether or not to compress the final CSV file into a ZIP archive.
- `temp-folder`: Folder in which to store exported files on the application server, whilst streaming results from the platform (and before they are moved to their final destination). This can either be an absolute file path or relative to the application server root. If omitted, OS-specific temporary file folders are used.
- `redirect-path`: Path to redirect user after export has been initiated.
- `folder`: Folder in which to store exported files on the application server. This can either be an absolute file path or relative to the application server root. The path can be written as an expression, with variables that are resolved for the user that is performing the export:
  - `{{id}}`: The ID of the user.
  - `{{name}}`: The user name of the user (if different from the ID).
  - `{{domain}}`: The domain of the user. For example, `/var/data/export/{{domain}}/{{id}}/`.

When an export has been completed, the system sends out a `twigkit.media.export.ExportCompleted` event. You can write your own event handler like so:

```
import com.google.common.eventbus.Subscribe; public class ExportListener {
@Subscribe public void export(ExportCompleted event) { System.out.println("Done
exporting " + event.getFile().getAbsolutePath() + " for user " + event.getUser());
// do something specific with this information } }
```

To register an event listener, add this entry to your application's `AppkitModule`:

```
@Override protected void configure() { ... eventBus.register(new ExportListener());
... }
```

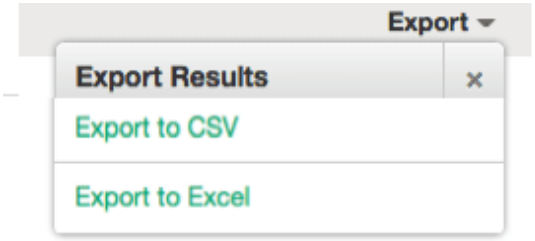
Depending on your business requirements, you might want to email the user a link to the exported CSV, list available CSVs on a home screen, and so forth.

Configuring export links within your search controls

Often it is useful to provide export options alongside your search results. You can achieve this using the above export tags along with some widget popover code:

```
<block:controls> <widget:popover id="export-controls" title="Export results" link-
text="Export" align="right" close-on-click="li"> <ul class="tk-stl-toggle-menu">
<li> <media:csv conf="platforms.gsa" query="query" delimiter="|" fetch="100000"
batch-size="100" fields="title,foo,bar">Export to CSV</media:csv></li> <li>
<media:excel conf="platforms.gsa" query="query" fetch="10000" fields="title,foo"
header="myTitle,myFoo">Export to Excel</media:excel></li> </ul> </widget:popover>
</block:controls>
```

This displays the various export tags in a list, which is shown on toggle of the Export drop-down link.



Edit the `link-text="Export"` to change the title of the link, and add further `<li>` elements with links in as required.

Image Formatting and Transcoding

The Image tag displays images from a given field in a result. It will automatically resize, and cache the image for performance.

Usage

This example shows how to display an image from a field:

```
<media:image field-name="image" field-prefix="http://foo.com" validate="true"
width="150"></media:image>
```

In this case, the image URL is retrieved from a field named `image`. Before rendering, the image URL is prefixed with http://foo.com *(http://foo.com)* and its existence is validated. It is then resized to 150 pixels wide.

To see all related attributes, see the *media:image* tag documentation.

Workflow and Data Processing Module

A workflow processor is an atomic unit of business logic that can modify a query before it is submitted to a search platform, or transform a search response before rendering. This capability is used extensively for doing runtime data cleaning when, for example, it is not feasible to re-index a whole collection to fix issues in the source data.

The workflow module ships with a number of query and response processors for common tasks out of the box and it is easy to hook these or your own ones into the query/response lifecycle. These include, but are not restricted to:

- **Query pattern matching.** Intercept queries for common terms that have no corresponding documents in the index (for example, postal codes, phone numbers, or email addresses).
- **Natural language processing (NLP).** Analyse free-text input typed in by the end-user to provide a structured, more specific search command to the backend engine. For example, given the query "restaurants in new york" the NLP pre-processor might produce a structured query along the lines of `category:restaurant and city:nyc`. Appkit has integration with a number of third-party NLP parsers, including Expert Systems, WolframAlpha and Smartlogic. More commonly, it is also easy to hook in custom natural language parsers, customized for a particular domain.
- **Data augmentation.** Transform and extend existing data on the fly with re-indexing, for example by fetching external linkages (look up latest stock ticker price, perform relational joins, etc.) or perform translation or lookups (replace ontology terms with common name).
- **Remove facet filters by pattern.** The workflow module includes a comprehensive set of runtime data cleansing tools. When re-indexing is not feasible Appkit can remove irrelevant filters from dynamic navigation options based on for example, regular expression patterns.

The figure below illustrates the query-response lifecycle including a pipeline of workflow processing.

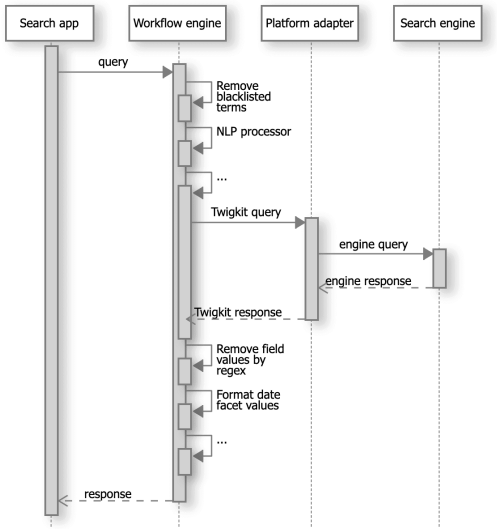


Figure 1. Sample query-response lifecycle involving the workflow engine.

In this example:

1. A search request is pre-processed before it is translated into an engine-specific command and sent to the search engine (for example, by removing specific forbidden or blacklisted words, applying natural language processing to the free-text portion of the query, etc.).
2. The data that comes back from the search engine is translated into a generic Appkit search response by the platform adapter, and then fed into a post-processing pipeline (which might, for example, apply regular expressions to remove specific terms from data fields, apply date formatting to facet aggregation values, etc.).
3. This produces finally a generic Appkit response that is returned to the search application, for rendering or further processing.

Workflow processors: building blocks of data transformation

An Appkit workflow processor is a Java class that is invoked and passed a reference to either a query or response object. A query pre-processor rewrites a query before it is submitted to the underlying search platform. Conversely, a response post-processor transforms a search response (both search results and facets) after it gets returned from a search engine (and before rendering). Workflow processors are typically declared in markup using JSP tags. Multiple processor tags can be specified for sequential processing of queries or responses.

Configuring a workflow pipeline

A workflow pipeline is configured at a platform level. This is useful for example for when you want to reuse processors in several places. This can be done using *configuration files*, chaining processors together along with a platform as a "workflow platform", which then can be referenced like any other platform.

Lucidworks has packaged up a selection of *commonly used response post-processors* for reuse.

Response Processors

The articles in this section describe response processors.

Field Processors

Field response processors include ones to perform these operations:

- Capitalize the display value of the given field names.
- Add Highlighting to Fields.
- Tag a document with classifications based on field values.
- Format a date object, replacing the original date value with another.
- Parse dates out of field values.
- Set the value of a field that is missing a value, based on the value of a different field.
- Parse the String value of the specified fields into an Object representation.
- Create a multivalued field from a single field value by using a separator.
- Extract the hostname from URLs and place it in a field named 'site'.
- Process fully qualified URLs in field values and markup, and add anchor tags for active links in the display values.
- Statically add metadata to documents that match a given regular expression.
- Replace field values (actual, display, or both) that are HTML or URL encoded with decoded values.
- Replace field values (actual, display, or both) that match a given regular expression with a different value.
- Make Twitter users and hashtags clickable in the display value.
- Duplicate a field, creating two separate instances.
- Create a new field by joining multiple existing fields using a pattern expression.
- Localize the values of a field using a specified bundle.

Filter Processors

Filter response processors include ones to perform these operations:

- Capitalize the display value of the given filter.
- Parse dates out of facet filter values.
- Sort facet filters by count.
- Sort facet filters by actual value.
- Replace display values in FacetFilters for specified or all Facets.
- Retrieve other filter values for the same facet, despite there being one or more filters for that facet already applied.
- Remove facet filters that match a regular expression pattern from a facet.
- Replace filter values (actual, display, or both) that match a given regular expression with a different value.
- (Since 4.2.0) Dynamically format the display value of a date range facet according to the size of the range.

Facet Processors

Facet response processors include ones to perform these operations:

- Duplicate a facet.
- Create facets.
- Fill in date filters for a facet.
- Parse and formatting filter values from one Date format to another.
- Create a pseudo date facet.
- Set or changing Parameters for a facet.
- Sort facets based on various properties and parameters.
- Process facet filter values formatted as paths to represent them as hierarchical trees.
- Remove facets from a Response.
- Create pseudo facets based on values in a given field or fields.
- Process hierarchical facets so that only one level is rendered at a time.
- Define a pseudo facet from a list of facets in the response.

General Response Processors

General response processors include ones to perform these operations:

- Perform a left join on fields.
- Log response metadata and attributes to log4j.
- Group results by a given Field value.

Facet Processors

Facet response processors include ones to perform these operations:

- Duplicate a facet.
- Create facets.
- Fill in date filters for a facet.
- Parse and formatting filter values from one Date format to another.
- Create a pseudo date facet.
- Set or changing Parameters for a facet.
- Sort facets based on various properties and parameters.
- Process facet filter values formatted as paths to represent them as hierarchical trees.
- Remove facets from a Response.
- Create pseudo facets based on values in a given field or fields.
- Process hierarchical facets so that only one level is rendered at a time.
- Define a pseudo facet from a list of facets in the response.

For more information about structuring hierarchical facet data in Fusion, see *Indexing Hierarchical Taxonomies*.

twigkit.search.processors.response.CopyFacetProcessor

Duplicate a facet, creating two separate instances.

`from (java.lang.String)`
Name of facet to copy (clone).

`to (java.lang.String)`
Name to assign to the new Facet.

twigkit.search.processors.response.CreateFacetProcessor

Create facets by name if they do not exist.

`facetNames (java.lang.String)`
Comma-separated list of facets that should be affected by this processor.

twigkit.search.processors.response.DateFacetIntervalProcessor

Processor for filling in date filters for a facet, based on a particular step unit (year, month, day). It assumes the named facet has date filters.

`facetNames (java.lang.String)`
Comma-separated list of facets that should be affected by this processor.

`inputFormat (java.lang.String)`
Pattern for the original date to be parsed.

`outputFormat (java.lang.String)`
Format of the output.

`step (java.lang.String)`
`year`, `month`, or `day`

twigkit.search.processors.response.DateFacetValueFormatter

Parse and format filter values from one Date format to another.

`facetNames (java.lang.String)`
Comma-separated list of facets that should be affected by this processor.

`sourceFormat (java.lang.String)`
Pattern for the original date to be parsed.

`format (java.lang.String)`
Format of the output.

twigkit.search.processors.response.DateRangeFacetProcessor

Create a pseudo date facet.

`field (java.lang.String)`
The name of the field containing the Date value.

`displayName (java.lang.String)`
The display name of the Facet generated for field.

twigkit.search.processors.response.FacetParameterProcessor

Set or change Parameters for a facet.

`operation (java.lang.String)`
Currently only supports 'set'.

`parameter (java.lang.String)` The name of the parameter to set.

`value (java.lang.Object)`
The value to set the parameter to.

twigkit.search.processors.response.FacetSortProcessor

Sort facets based on properties and parameters.

`sort (java.lang.String)`
Sort string as per Query parameter (`-field1,field2,-field3`).

twigkit.search.processors.response.PathHierarchicalProcessor

Process facet filter values formatted as paths to represent them as hierarchical trees, for example, "Europe/Central Europe/Germany/Berlin".

`facetNames (java.lang.String)`
Comma-separated list of facets that should be affected by this processor.

`levelSeparator (java.lang.String)`
The character sequence that represents a level, for example, levelSeparator="/".

	After this processor has been applied to a facet, the facet filters are sorted alphabetically by value (rather than by count).

twigkit.search.processors.response.RemoveFacetsProcessor

Remove facets from a Response by name.

`facetNames (java.lang.String)`
Comma-separated list of facets that should be affected by this processor.

twigkit.search.processors.response.ShallowFacetGenerator

Create pseudo facets based on values in a given field or fields.

`fields (java.lang.String)`
Comma-separated list of fields that should be affected by this processor.

twigkit.search.processors.response.ShallowHierarchicalFacetProcessor

Process hierarchical facets so that only one level is rendered at a time, with an HTTP request/response between levels.

`facetNames (java.lang.String)`
Comma-separated list of facets that should be affected by this processor.

twigkit.search.processors.response.SequentialFacetChoiceProcessor

Define a pseudo facet from a list of facets in the response. The facet that gets copied over to the new pseudo facet will be determined by looking at a sequence of facets (specified by the developer) and choosing the first one that does not have any filter constraints against it. When the current query contains filters against all facets in the sequence, the last facet in the sequence is copied to the pseudo facet.

For example, assume you have fields in your schema for `Region`, `Country`, and `City` that you want to facet on to guide users in their search. These facets forms a natural hierarchy of `Region > Country > City`. Instead of giving the user a choice from all of the three facets at every step (which would show for example, cities from all regions mixed together), you want to present the user with a single `Location` facet that moves from `Region` to `Country` and finally `City` as users make more specific choices.

To illustrate this, assume you have these data values across your corpus of documents:

```
- Region: Europe, America, Asia - Country: France,United Kingdom,Germany, Mexico,
Brazil,Japan,Korea - City:
Paris,Lyon,London,Manchester,Berlin,Hamburg,Mexico,Tijuana,Sao
Paulo,Brazilia,Tokyo,Kyoto,Seoul
```

Configure a `SequentialFacetChoiceProcessor` with these parameters:

```
facetNames: Region,Country,City target: Location
```

In your search application, present the user with a *single* facet - `Location`. Now consider:

- If no `Location` filter is specified, then the `Location` facet will contain `Region` values: Europe, America, Asia.
- If the user chooses `Asia` as a location, then the `Location` facet will contain the follow `Country` option: Japan, Korea.
- Finally, if the user chooses `Japan`, then the `Location` facet will contain these options: Tokyo, Kyoto.

`facetSequence (java.lang.String)`
Comma-separated list of facets that should be copied to the new target facet.

`target (java.lang.String)`
Name to assign to the new facet.

`separator (java.lang.String)`
Character sequence to separate between facet names when assembling a display label for the new target facet. By default this is ` ▸ ` (an arrow pointing to the right).

Field Processors

Field response processors include ones to perform these operations:

- Capitalise the display value of the given field names.
- Add Highlighting to Fields.
- Tag a document with classifications based on field values.
- Format a date object, replacing the original date value with another.
- Parse dates out of field values.
- Set the value of a field that is missing a value, based on the value of a different field.
- Parse the String value of the specified fields into an Object representation.
- Create a multivalued field from a single field value by using a separator.
- Extract the hostname from URLs and place it in a field named 'site'.
- Process fully qualified URLs in field values and markup, and add anchor tags for active links in the display values.
- Statically add metadata to documents that match a given regular expression.
- Replace field values (actual, display, or both) that are HTML or URL encoded with decoded values.
- Replace field values (actual, display, or both) that match a given regular expression with a different value.
- Make Twitter users and hashtags clickable in the display value.
- Duplicate a field, creating two separate instances.
- Create a new field by joining multiple existing fields using a pattern expression.
- Localize the values of a field using a specified bundle.

twigkit.search.processors.response.CapitaliseFieldValuesProcessor

Capitalise the display value of the given field names.

```
name: twigkit.search.processors.response.CapitaliseFieldValuesProcessor fields:
firstName,lastName
```

`fields (java.lang.String)`
Comma-separated list of fields that should be affected by this processor.

twigkit.search.processors.response.HighlightFieldValuesProcessor

Add Highlighting to Fields. For a more detailed overview, see *the highlighting page*.

twigkit.search.processors.response.FieldEntityExtractor

Tag a document with classifications based on field values. Using the specified fields, look for patterns provided in a properties file and add classifications to a given field if the value matches.

```
name: twigkit.search.processors.response.FieldEntityExtractor fields: issues
classificationField: categorizedIssues bundle: my-issues
```

With:

```
#my-issues_en.properties in /resources/conf Foo Bar Bam
```

This can be used to pull out keywords from a given field into a new field for display. If the Example above had a field named 'issues' with 'The problem is Foo something else' as value, then the FieldEntityExtractor would match on Foo, and stick 'Foo' into a new categorizedIssues field. This can be used to create new metadata fields, or even filter fields with sensitive data, to just pull out what you want to show.

`bundle (java.lang.String)`
A properties file containing entities to extract and an optional replacement value when found (for example, IBM = International Business Machines where the latter would be stored as the match for IBM). These are expressed as regular expressions (regex) so as well as simple matching based on whether the value was found within a field, entities can be recognized based on a specific pattern within the text of the chosen fields.

`fields (java.lang.String)`
Comma-separated list of fields that should be used to search for matches in order to classify the document.

`classificationField (java.lang.String)`
The field in which to store the classification values. The flexible pattern based approach used allows a document to be tagged with multiple classifications if several matches are found.

twigkit.search.processors.response.DatePartExtractor

Format a date object, replacing the original date value with another. *Use on fields that are already date objects*.

```
name: twigkit.search.processors.response.DatePartExtractor fields: issues pattern: dd MMM yyyy
```

`fields (java.lang.String)`
Comma-separated list of fields that should be affected by this processor.

`pattern (java.lang.String)`
Format a Date object according to this Date pattern. Use Java's SimpleDateFormat *(http://docs.oracle.com/javase/7/docs/api/java/text/SimpleDateFormat.html)* syntax. In the examples, for example, you will see that `pattern="EEE, MMM d, ''yy"` results in `Wed, Jul 4, '01`.

twigkit.search.processors.response.FieldDateParser

Parse dates out of field values. To convert String data into Date objects.

```
name: twigkit.search.processors.response.FieldDateParser fields: issues pattern: dd MMM yyyy
```

`fields (java.lang.String)`
Comma-separated list of fields that should be affected by this processor.

`pattern (java.lang.String)`
The pattern to use when parsing the String value to a Date.

twigkit.search.processors.response.FallbackFieldValue

Set the value of a field that is missing a value, based on the value of a different field.

```
name: twigkit.search.processors.response.FallbackFieldValue field: phoneNumber fallback: mainPhoneNumber pattern: values: display decode: false
```

`field (java.lang.String)`
Field that should be affected by this processor.

`fallback (java.lang.String)`
Field to use for the fallback values.

`pattern (java.lang.String)`
Regex pattern to use to extract a value from the fallback field

`values (java.lang.String)`
Which forms of the value to check for emptiness - 'display', 'actual', or 'either'

`decode (java.lang.Boolean)`
Whether to perform URL decoding on the fallback values

twigkit.search.processors.response.FieldValueParser

Parse the String value of the specified fields into an Object representation.

```
name: twigkit.search.processors.response.FieldValueParser fields: amount
```

`fields (java.lang.String)`
Comma-separated list of fields that should be affected by this processor.

twigkit.search.processors.response.FieldToMultiValueFieldProcessor

Create a multivalued field from a single field value by using a separator.

```
name: twigkit.search.processors.response.FieldToMultiValueFieldProcessor field:
cities separator: ,
```

`field (java.lang.String)`
The name of the field containing the value to be split.

`separator (java.lang.String)`
The separator to use when splitting the value.

twigkit.search.processors.response.HostNameExtractor

Extract the hostname from URLs and place it in a field named 'site'.

```
name: twigkit.search.processors.response.HostNameExtractor
```

`fields (java.lang.String)`
Comma-separated list of fields that should be affected by this processor.

twigkit.search.processors.response.LinkMarkupProcessor

Process fully qualified URLs in field’s 'actual' value, and markup with anchor tags for active links in display value.

```
name: twigkit.search.processors.response.LinkMarkupProcessor fields: url
```

`fields (java.lang.String)`
Comma-separated list of fields that should be affected by this processor.

twigkit.search.processors.response.RegExFieldValueTaggerProcessor

Statically add metadata to documents that match a given regular expression (in one or more fields).

```
name: twigkit.search.processors.response.RegExFieldValueTaggerProcessor fields:
path classificationField: type pattern: [\w]+
```

The example above looks at a field named path for example, "foo/bar/bam", and breaks all words into a multivalued field named type with multiple values foo, bar and bam using a regex pattern.

If you want to change data rather than just use it, use a ReplaceFieldValue processor.

`fields (java.lang.String)`
Comma-separated list of fields that should be affected by this processor.

`classificationField (java.lang.String)`
The field that should contain the metadata classificationValue if the pattern matches.

`pattern (java.lang.String)`
The pattern to match to the values in the fields defined with the fields parameter.

twigkit.search.processors.response.DecodeFieldValueProcessor

Replace field values (actual, display, or both) that are HTML or URL encoded with decoded values. A use case might be to replace the display value of a URL field. Example usage:

```
name: twigkit.search.processors.response.DecodeFieldValueProcessor fields:
url_display encoding: url values: display
```

`fields (java.lang.String)`
Comma-separated list of fields that should be affected by this processor.

`values (java.lang.String)`
Whether to replace 'actual', 'display' or 'both' values.
Default: 'both'

`encoding (java.lang.String)`
The encoding of the value to be decoded; 'url' or 'html'.
Default: 'url'

twigkit.search.processors.response.ReplaceFieldValue

Replace field values (actual, display, or both) that match a given regular expression with a different value. The replacement value can contain back-references to matches. A common use case for this is to use a CopyFieldProcessor first, then make changes. Example:

```
name: twigkit.search.processors.response.ReplaceFieldValue fields: folder replace:
^(.*/).*$ with: $1
```

The example above strips off a file name from the end of a folder path to leave just the path using capture expressions and back-references regular expressions *(http://www.regexplanet.com/advanced/java/index.html)*.

`fields (java.lang.String)`
Comma-separated list of fields that should be affected by this processor.

`replace (java.lang.String)`
The pattern to replace (can contain regular expressions).

`with (java.lang.String)`
The replacement (can contain backreferences to the regular expression pattern).

`values (java.lang.String)`
Whether to replace 'actual', 'display' or 'both' values. Default: 'both'

`ignoreCase (java.lang.Boolean)`
Whether to ignore case during pattern matching. Default: false

twigkit.search.processors.response.TweetMarkupProcessor

Make Twitter users and hashtags clickable in the display value.

```
name: twigkit.search.processors.response.TweetMarkupProcessor fields: twitter_user
```

`fields (java.lang.String)`
Comma-separated list of fields that should be affected by this processor.

twigkit.search.processors.response.CopyFieldProcessor

Duplicate a field, creating two separate instances.

```
name: twigkit.search.processors.response.CopyFieldProcessor from: url to: my_url
```

`from (java.lang.String)`
Name of field to copy (clone).

`to (java.lang.String)`
Name to assign to the new field.

twigkit.search.processors.response.ConcatenateFields

Create a new field by joining multiple existing fields using a pattern expression.

`expression (java.lang.String)`
Concatenated field pattern defined using double curly braces (see below)

`target (java.lang.String)`
Name of the new field to create in the response

```
name: twigkit.search.processors.response.ConcatenateFields expression:
{{MemberStreet1}} {{MemberStreet2}} {{MemberCity}} {{MemberState}}
{{MemberZipCode}} {{MemberCountry}} target: compositeAddress
```

Or create a new image field that uses a field value as part of an expression:

```
name: twigkit.search.processors.response.ConcatenateFields expression:
http://your/custom/path/{{MemberCity}}.jpg target: image_url
```

Then use a `<media:image>` tag to output the image field in the result output:

```
<media:image field-name="image_url" width="156" height="156" ... >
```

twigkit.search.processors.response.LocaliseFieldValueProcessor

Localize the values of a field using a specified bundle. For example:

```
name: twigkit.search.processors.response.LocaliseFieldValueProcessor bundle:
languages locale: en fields: language
```

As an example, add a file named languages_en.properties to your class path (for example, to src/main/resources) and containing these key-value pairs (truncated):

```
aa = Afrikaans ab = Abkhaz am = Amharic ar = Arabic az = Azerbaijani ba = Bashkir
be = Belarusian bg = Bulgarian bm = Bamanankan bn = Bengali bo = Tibetan br =
Breton bs = Bosnian ca = Catalan co = Corsican cr = Cree cs = Czech cy = Welsh etc.
```

`fields (java.lang.String)`
Comma-separated list of fields that should be affected by this processor.

`values (java.lang.String)`
Whether to replace 'actual', 'display' or 'both' values. Default: `both`

`bundle (java.lang.String)`
The bundle to use.

`locale(java.lang.String)`
The locale to use.

Filter Processors

Filter response processors include ones to perform these operations:

- Capitalize the display value of the given filter.
- Parse dates out of facet filter values.
- Sort facet filters by count.
- Sort facet filters by actual value.
- Replace display values in FacetFilters for specified or all Facets.
- Retrieve other filter values for the same facet, despite there being one or more filters for that facet already applied.
- Remove facet filters that match a regular expression pattern from a facet.
- Replace filter values (actual, display, or both) that match a given regular expression with a different value.
- (Since 4.2.0) Dynamically format the display value of a date range facet according to the size of the range.

twigkit.search.processors.response.CapitaliseFilterValuesProcessor

Capitalise the display value of the given filter.

`facetNames (java.lang.String)`
Comma-separated list of facets that should be affected by this processor.

twigkit.search.processors.response.FacetFilterDateParser

Parse dates out of facet filter values (which converts the underlying value type from String to Date).

`facetNames (java.lang.String)`
Comma-separated list of facets that should be affected by this processor.

`pattern (java.lang.String)`
Date formatting pattern of the String to be parsed for example, dd/MM/yyyy

twigkit.search.processors.response.FacetFilterSortByCount

Sort facet filters by count.

`facetNames (java.lang.String)`
Comma-separated list of facets that should be affected by this processor.

twigkit.search.processors.response.FacetFilterSortByValue

Sort facet filters by actual value.

`facetNames (java.lang.String)`
Comma-separated list of facets that should be affected by this processor.

`type (java.lang.String)`
Sort by `number`, `string` or `date`.
Default: `string`

`direction (java.lang.String)`
Either 'ascending' or 'descending'.

twigkit.search.processors.response.FacetFilterValueMappingProcessor

Replace display values in FacetFilters for specified or all Facets. This is a convenient way, for example, to replace display values in facets with boolean values.

`facetNames (java.lang.String)`
Comma-separated list of facets that should be affected by this processor.

`mappings (java.lang.String)`
Mappings parameter defines the mappings/replacements that should be done on the FacetFilter display values. For example, mappings="en=English,fr=French" will replace all instances of en and fr in the FacetFilter list with the full English/French variants.

twigkit.search.processors.response.ParallelFilterProcessor

Retrieve other filter values for the same facet, despite there being one or more filters for that facet already applied.

`facetNames (java.lang.String)`
Comma-separated list of facets that should be affected by this processor.

`platform (twigkit.platform.Platform)`
The platform instance to use when resubmitting the query with the facet filters removed (so that parallel filter values can be retrieved).

twigkit.search.processors.response.RemoveFacetFiltersByRegex

Remove facet filters that match a regular expression pattern from a facet.

`facetNames (java.lang.String)`
Comma-separated list of facets that should be affected by this processor.

`pattern (java.lang.String)`
A regular expression pattern for the filters that should be removed.

twigkit.search.processors.response.ReplaceFilterValue

Replace filter values (actual, display, or both) that match a given regular expression with a different value. The replacement value can contain back references to matches.

`facetNames (java.lang.String)`
Comma-separated list of facets that should be affected by this processor.

`replace (java.lang.String)`
The pattern to replace (can contain regular expressions).

`with (java.lang.String)`
The replacement (can contain backreferences to the regular expression pattern).

`values (java.lang.String)`
Whether to replace 'actual', 'display' or 'both' values.
Default: 'both'

`ignoreCase (java.lang.Boolean)`
Whether to ignore case during pattern matching.
Default: false

twigkit.search.processors.response.ApplyDynamicSolrDateFormattingResponseProcessor

Since 4.2.0

Dynamically format the display value of a date range facet according to the size of the range. This assumes the facet was generated by applying the `twigkit.search.processors.query.DynamicSolrDateRangeQueryProcessor` processor.

`facetNames (java.lang.String)`
Comma-separated list of facets that should be affected by this processor.

`day-format (java.lang.String)`
The format to apply to day size range filters.
Default: `dd-MM-yyyy`

`month-format (java.lang.String)`
The format to apply to month size range filters.
Default: `MM-yyyy`

`year-format (java.lang.String)`
The format to apply to year size range filters.
Default: `yyyy`

Example configuration

```
name:
twigkit.search.processors.response.ApplyDynamicSolrDateFormattingResponseProcessor
facet-names: Modified Date month-format: yyyy-MM day-format: yyyy-MM-dd
```

General Response Processors

General response processors include ones to perform these operations:

- Perform a left join on fields.
- Log response metadata and attributes to log4j.
- Group results by a given Field value.

twigkit.search.processors.response.LeftJoinProcessor

Perform a left join on the specified fields, adding results with the same values in the specified fields as 'related' results. Appkit carries out additional searches against the platform to retrieve each related result.

`joinFields (java.lang.String)`
Comma-separated list of fields that should be affected by this processor.

twigkit.search.processors.response.ResponseLoggingProcessor

Log Response metadata and attributes to log4j as TRACE.

twigkit.search.processors.response.ResultGroupProcessor

Group results by a given Field value, making group entries related results for the first one.

`groupBy (java.lang.String)`
The field containing the identifying value to group results by.

Query Processors

twigkit.search.processors.query.AddFilterProcessor

Add a Filter to a Query. By creating or passing in a Filter instance, you can override or set all attributes of the filter that will then be added to the Query.

Parameters

`filter (twigkit.model.Filter)`
Instance of a filter to add (optional).

`field (java.lang.String)`
Field name of the filter.

`actualValue (java.lang.String)`
The actual value of the filter to be applied.

`displayValue (java.lang.String)`
The display value of the filter to be applied.

`hidden (java.lang.Boolean)`
Whether the filter should be hidden (visible in breadcrumbs and URLs) or not.

`exclude (java.lang.Boolean)`
Whether the filter should exclude values in the given field.

twigkit.search.processors.query.CompositeFieldProcessor

Create a pseudo composite field for user fielded search, where a users query expression will be reformatted as an OR query on a set of fields.

Parameters

`fieldNames (java.lang.String)`
Names of fields to apply query terms to as an OR query.

`compositeField (java.lang.String)`
Name of the pseudo composite field that the user can invoke via fielded search syntax (compositeField:expression).

twigkit.search.processors.query.ReplaceFilterValue

Replace all Query filter values (actual, display or both) that match a given regular expression with a different value. The replacement value can contain back-references to matches.

Parameters

`replace (java.lang.String)`
The pattern to replace (can contain regular expressions).

`with (java.lang.String)`
The replacement (can contain backreferences to the regular expression pattern).

`values (java.lang.String)`
Whether to replace 'actual', 'display' or 'both' values. Default: 'both'

`ignoreCase (java.lang.Boolean)`
Whether to ignore case during pattern matching.
Default: false

twigkit.search.processors.query.QueryLoggingProcessor

Logs the full details of each query sent to the platform at TRACE log level.

twigkit.social.processor.AddBookmarkListToQueryProcessor

Query processor for retrieving a list of bookmarks in a query

Parameter

`id (java.lang.String)`
Required. The field name to use as a filter when filtering the query with the bookmark `id`.

Example configuration

```
name: twigkit.social.processor.AddBookmarkListToQueryProcessor id: ID
```

> There is no reliable way of making the query return nothing if there are no bookmarks in the list, so this logic is best handled in the JSP, testing whether they exist using a conditional test with a `social:getBookmarkCount` function before applying the processor.

twigkit.search.processors.query.DynamicSolrDateRangeQueryProcessor

Since 4.2.0

Query processor to create Solr (and Fusion) query parameters to enable date range faceting with a dynamic bucket size based on the current filters applied to the query. This processor should be used in conjunction with the `twigkit.search.processors.response.ApplyDynamicSolrDateFormattingResponseProcessor` response processor.

Parameters

`fields (java.lang.String)`
Required
Comma separated list of fields to apply the processor to. If the field is aliased on the platform it must be provided as `field=alias`.

`max-data-points (java.lang.int)`
Required
The number of date buckets do you want to generate at a maximum. This drives the size of the buckets being requested. For example if the overall interval is 1 year, and this parameter is set to 36, the size of the buckets will be 1 month. If this parameter is set to 500 the size will be 1 day.

`dynamic-buckets (java.lang.Boolean)`
Optional
Whether or not to apply the `facet.range.gap` Solr query parameter or rely on it being set elsewhere.
Default: `true`

`default-start-date (java.lang.Date)`
_Required+
When no date filters are applied on the query, this will be the lower bound of the first bucket requested to the platform endpoint.

`default-end-date (java.lang.Date)`
Required
When no date filters are applied on the query, this will be the higher bound of the last bucket requested to the platform endpoint.

`default-other (java.lang.String)`
Optional
This specified the solr `facet.range.other` parameter. If none is specified this parameter will not be set on the query. See here *(https://lucene.apache.org/solr/guide/7_2/faceting.html#range-faceting)* for the different values this parameter can take.

`default-unit (java.lang.String)`
Optional
Initial size of the buckets to request when no filters are applied. Default: one year

Example configuration

```
name: twigkit.search.processors.query.DynamicSolrDateRangeQueryProcessor fields: last_modified_dateX=Modified Date max-data-points: 36 default-start-date: 2016-01-01T00:00:00.000Z default-end-date: 2018-06-01T00:00:00.000Z
```

Social and Collaboration Tools

The social and collaboration tools provided by the Social module extend the search application with the ability to store user-generated information on individual documents and document collections, and to share this information among groups of authenticated users.

Features include:

- **Saved searches.** Users can save important keyword search queries, apply filters, and retrieve the information later.
- **Saved results.** Users can bookmark results, add them to a basket, and search those saved items.
- **Rating.** Users can choose to rate or "like" results, giving a social signal to their peers. Rating analytics can be used to influence ranking.
- **Tagging.** Users can tag content with keywords, which content owners can use to improve data quality.
- **Annotations.** Users can annotate/comment on results and edit or delete their own comments. Users cannot edit or delete the comments of others.
- **Topics.** Users can create topics and any captured metadata. For example, saved searches can be associated with a topic.

User-generated content can be stored in a relational database or any other kind of data store, using our pluggable architecture.

Authentication and authorization of users is managed via the *Security module*.

Terminology

While much of the language used to describe collaborative tools is carefully chosen to be self explanatory, there are several general concepts which are used throughout and are introduced in this section.

Annotations

Key to the collaboration enabled by inclusion of this module, there are several types of *annotation* which can be applied to documents. These include bookmarks, comments, and likes, which are commonly found in many applications and provide a versatile way for users to contribute and save information during their search.

Topics

A *topic* is a general term for a subject that users are intending to collect information about. This module provides facilities for users to create a named group with which they can associate any number of annotated documents. Topics themselves can be created and selected using the *Topic Form tag*. Many of the collaborative tags included in this module have a *topic* attribute that links the saved information to a specific topic (which can be picked up from the URL). Topics can be defined by the developer as being 'namespaced' so they can be global (available for all users to collaborate on) or local (only viewable by the individual user that created them).

Collections

Some annotations can be categorized by *collection*. This is a mechanism for separating annotations of different types of document. There is a subtle difference between topics and collections, in that collections are usually determined by the categories of information in the data and generally pre-defined by the application developer offering a small number of collections with which to associate annotations. Collections are intended for a finite number of categorical groups whereas any number of topics can be created.

Social and Collaboration tags

Like Documents

The *collaborate:like* tag displays controls for the user to "like" a document. It also displays statistics on the number of times a specified target document has been liked.

Saving Queries

The Appkit social module lets users save and retrieve their search queries. Use the *collaborate:saved-query-list* tag to let users save and retrieve their saved search queries. Alternatively, you can enable social features in the *search:box* to have access to the same form.

Bookmarks

The *collaborate:bookmark* tag displays controls for adding a new bookmark, in addition to statistics on the number of times a specified target document has been bookmarked. This tag is typically used within a Result List tag, to enable the user to bookmark a particular document in the result list.

Bookmark List tag The *collaborate:bookmark-list tag* retrieves and displays all bookmarks that have been created by a particular user, are within a given topic, or are linked to a particular document (or any combination of the three constraints). Alternatively, you can enable social features in the *search:box* to have access to the same form.

Commenting

The Appkit Social module lets users comment on search results and display other users' comments. This gives users a flexible way to interact and contribute in an application, and can be used to implement features such as feedback, discussion, and reviewing systems.

The *collaborate:comment-form* and *collaborate:comment-list* tags are used respectively to create and retrieve these annotations.

Create Topics

The *collaborate:topic-form* tag lets users create named topics with which to associate saved information. Topics can be restricted to the user who creates a topic or global to all users. The *collaborate:topic-list* tag will display these topics.

App Studio Gallery

This gallery provides visual examples of App Studio elements, with code snippets that you can copy, paste, and edit in your own App Studio views.

Button

Default

Standard button

Example

```
<button styling="button">Follow</button>
```

Secondary

Button can be primary and secondary

Example

```
<button styling="button">Okay</button> <button styling="button button-
secondary">Cancel</button>
```

Icon

Can be only icon

Example

```
<button styling="button"> <widget:icon icon="heart"></widget:icon> </button>
```

Disabled

Example

```
<button styling="disabled button"> <i class="icon icon-head"></i> Disabled
</button>
```

Size

Button are in different sizes

Example

```
<button styling="xs button">Extra Small</button> <button styling="sm
button">Small</button> <button styling="md button">Medium</button> <button
styling="lg button">Large</button> <button styling="xl button">Extra Large</button>
```

Floated

Button can float to left or right

Example

```
<button styling="button pull-right">Right Floated</button> <button styling="button
pull-left">Left Floated</button>
```

Checkbox

Default

Example

```
<input styling="checkbox-switch" id="switch1" type="checkbox"> <label
for="switch1"></label>
```

Square

Example

```
<input styling="checkbox-switch" id="switch2" type="checkbox"> <label
styling="square" for="switch2"></label>
```

Flat

Example

```
<input styling="checkbox-switch" id="switch3" type="checkbox"> <label
styling="flat" for="switch3"></label>
```

Thin

Example

```
<input styling="checkbox-switch" id="switch4" type="checkbox"> <label
styling="thin" for="switch4"></label>
```

Colors

Example

```
<input styling="checkbox-switch" id="switch5" type="checkbox"> <label
styling="positive" for="switch5"></label> <input styling="checkbox-switch"
id="switch6" type="checkbox"> <label styling="negative" for="switch6"></label>
<input styling="checkbox-switch" id="switch8" type="checkbox"> <label
styling="warning" for="switch8"></label> <input styling="checkbox-switch"
id="switch10" type="checkbox"> <label styling="info" for="switch10"></label>
```

Size

Example

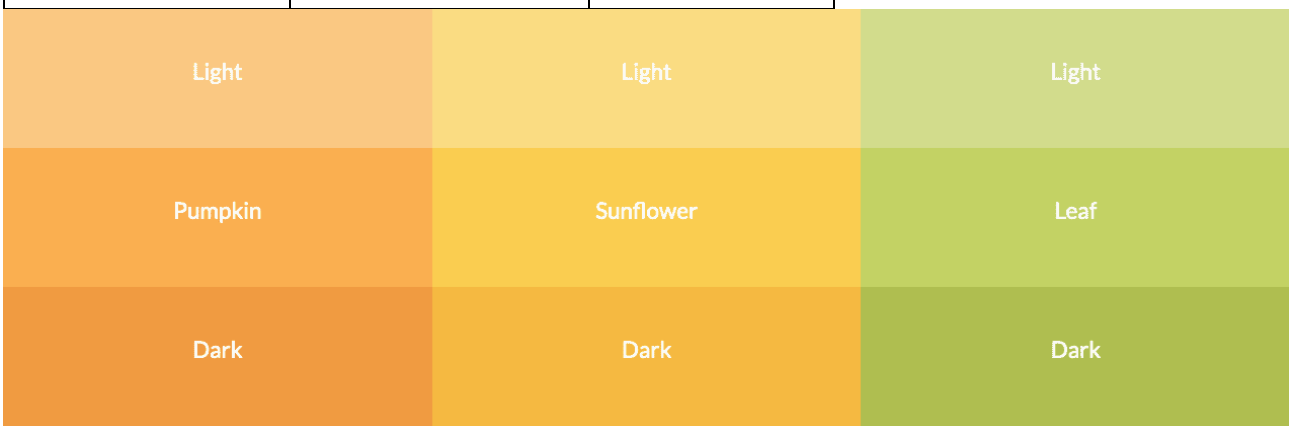
```
<input styling="checkbox-switch" id="switch13" type="checkbox"> <label
styling="tiny" for="switch13"></label> <input styling="checkbox-switch"
id="switch14" type="checkbox"> <label styling="massive" for="switch14"></label>
```

Color

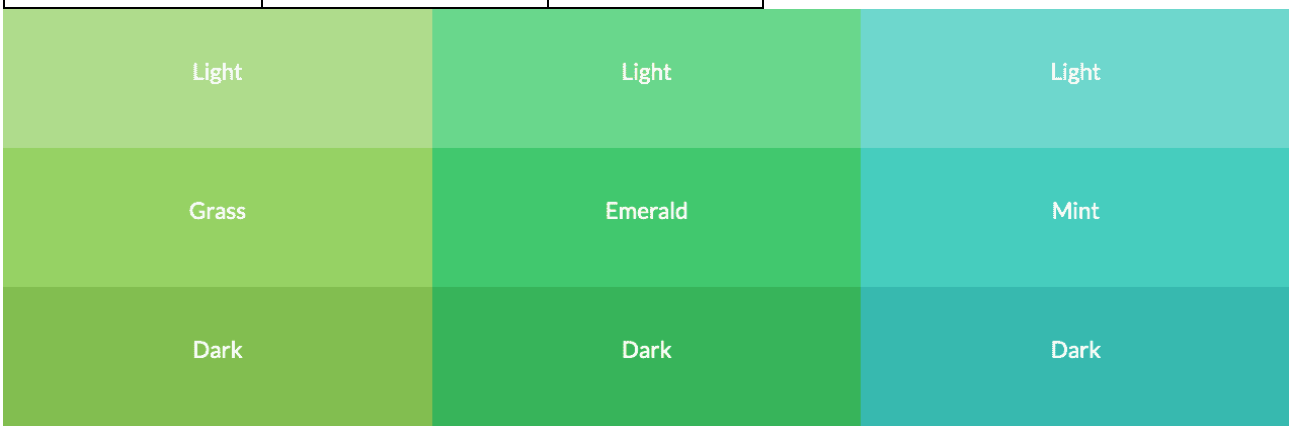
All of the colors can be used in the styling attribute: `styling="bg-raspberry text-paper-light"`

`bg-primary-light`	`bg-secondary-light`	`bg-tertiary-light`
`bg-primary`	`bg-secondary`	`bg-tertiary`
`bg-primary-dark`	`bg-secondary-dark`	`bg-tertiary-dark`

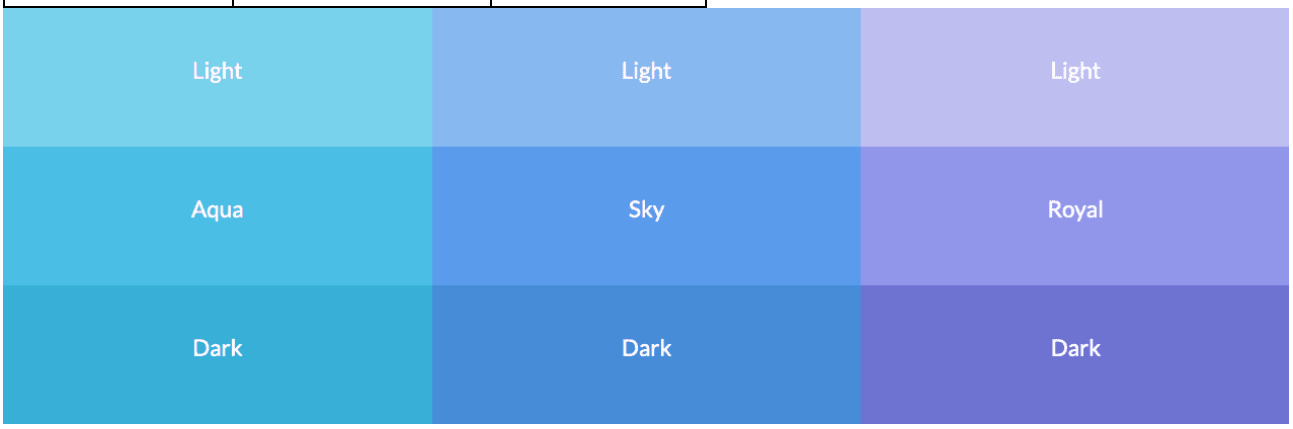
`bg-raspberry-light`	`bg-strawberry-light`	`bg-carrot-light`
`bg-raspberry`	`bg-strawberry`	`bg-carrot`
`bg-raspberry-dark`	`bg-strawberry-dark`	`bg-carrot-dark`



`bg-pumpkin-light`	`bg-sunflower-light`	`bg-leaf-light`
`bg-pumpkin`	`bg-sunflower`	`bg-leaf`
`bg-pumpkin-dark`	`bg-sunflower-dark`	`bg-leaf-dark`

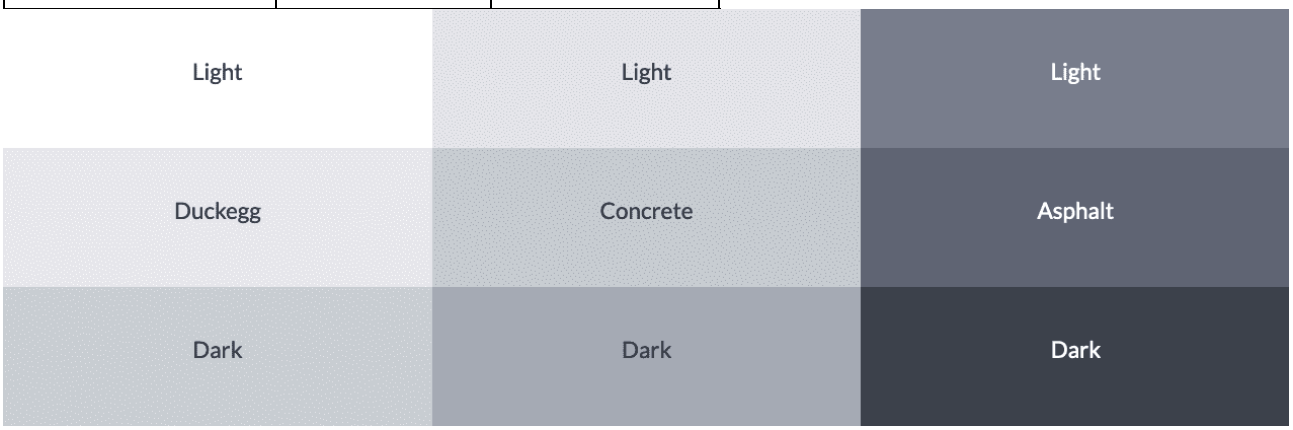


`bg-grass-light`	`bg-emerald-light`	`bg-mint-light`
`bg-grass`	`bg-emerald`	`bg-mint`
`bg-grass-dark`	`bg-emerald-dark`	`bg-mint-dark`



`bg-aqua-light`	`bg-sky-light`	`bg-royal-light`
`bg-aqua`	`bg-sky`	`bg-royal`
`bg-aqua-dark`	`bg-sky-dark`	`bg-royal-dark`

`bg-lavender-light`	`bg-pink-light`	`bg-paper-light`
`bg-lavender`	`bg-pink`	`bg-paper`
`bg-lavender-dark`	`bg-pink-dark`	`bg-paper-dark`



`bg-duckegg-light`	`bg-concrete-light`	`bg-asphalt-light`
`bg-duckegg`	`bg-concrete`	`bg-asphalt`
`bg-duckegg-dark`	`bg-concrete-dark`	`bg-asphalt-dark`

Comment

Default

Example

Stevie Feliciano 2 days ago 5 Faves
Hey guys, I hope this example comment is helping you read this documentation.

```
<div styling="comment-list"> <div styling="comment"> <a class="avatar"> <img
src="assets/avatars/j.jpg"> </a> <div class="content"> <a class="author">Stevie
Feliciano</a> <div class="metadata"> <div class="date">2 days ago</div> <div
class="rating"> <i class="star icon"></i> 5 Faves </div> </div> <div class="text">
Hey guys, I hope this example comment is helping you read this documentation.
</div> </div> </div> </div>
```

Comment List

Example

Paul 03/08/2016 at 5:42PM

Who doesn't love Police Academy?!

Reply Delete

a.

John 03/08/2016 at 5:42PM

Look at me! I'm nested!

Reply

a.

Ethan 03/08/2016 at 5:42PM

I'm nested even further.

Reply

Matt 03/08/2016 at 5:42PM

I have a default avatar and a long message.
Lorem ipsum dolor sit amet, consectetur adipisicing elit. Accusamus ad adipisci delectus dolore, dolorem facere facilis hic impedit iste laboriosam maiores nulla perspiciatis placeat praesentium qtka qtks repellat rerum ullam.

Reply

Steve 03/08/2016 at 5:42PM

Great job!

Reply

Reply...

Add Comment

```
<social:comment-list target="{string}" platform="{object}" [format="{string}" ]="" [delete-label="{string}" [show="{number}" [show-more="{number}" [show-more-label="{string}" [show-less-label="{string}" [empty-text="{string}" [default-delete-message="{string}" [avatars="{string}"> </social:comment-list>
```

Field

Example

Sir John Carew Eccles

year 1963

Gender Male Age 60 Facial Hair none

Country of birth Australia

Prize Category
medicine

"for their discoveries concerning the ionic mechanisms involved in excitation and inhibition in the peripheral and central portions of the nerve cell membrane"

```
<search:result result="response | result:5"> <!-- Title field styling-->
<search:field name="full_name" styling="title"></search:field> <!-- Default - label
left --> <search:field name="year"></search:field> <!-- Inline labels -->
<search:field name="gender" label="Gender" styling="label-inline"></search:field>
<search:field name="age" label="Age" styling="label-inline" show="1">
</search:field> <search:field name="facialhair" label="Facial Hair" styling="label-
inline"></search:field> <br> <!-- Label left that moves on top for smaller screens
--> <search:field name="Country of birth" styling="label-left"></search:field> <!--
Label is always above the field --> <search:field name="Prize Category"
styling="label-above"></search:field> <!-- Description styling, ideal for blocks of
text that don't require a label --> <search:field name="motivation"
styling="description"></search:field> </search:result>
```

Form

Default Compact Form

Example

A compact inline form (Default)

Email Password Remember me Sign in

```
<form styling="form"> <fieldset> <legend>A compact inline form (Default)</legend>
<input type="email" placeholder="Email"> <input type="password"
placeholder="Password"> <label for="remember"> <input id="remember"
type="checkbox"> Remember me </label> <button type="submit" styling="button">Sign
in</button> </fieldset> </form>
```

Stacked

Example

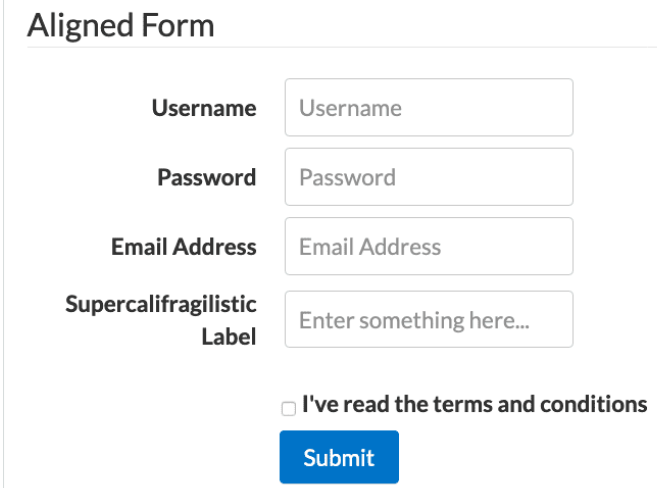
A Stacked Form

Email
Email
Password
Password
State
AL
Remember me
Sign in

```
<form styling="form form-stacked"> <fieldset> <legend>A Stacked Form</legend>
<label for="email">Email</label> <input id="email" type="email"
placeholder="Email"> <label for="password">Password</label> <input id="password"
type="password" placeholder="Password"> <label for="state">State</label> <select
id="state"> <option>AL</option> <option>CA</option> <option>IL</option> </select>
<label for="remember" class="pure-checkbox"> <input id="remember" type="checkbox">
Remember me </label> <button type="submit" styling="button">Sign in</button>
</fieldset> </form>
```

Aligned Form

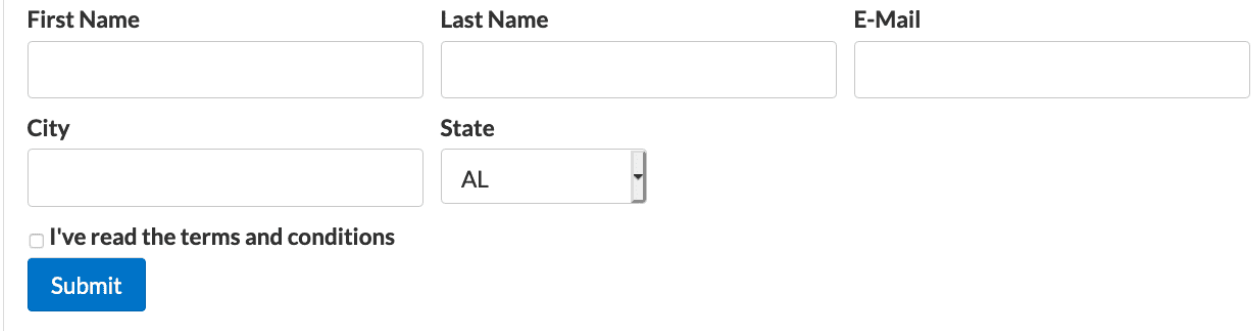
Example




```
<form styling="form form-aligned"> <fieldset> <legend>Aligned Form</legend> <div
styling="control-group"> <label for="name">Username</label> <input id="name"
type="text" placeholder="Username"> </div> <div styling="control-group"> <label
for="password">Password</label> <input id="password" type="password"
placeholder="Password"> </div> <div styling="control-group"> <label
for="email">Email Address</label> <input id="email" type="email" placeholder="Email
Address"> </div> <div styling="control-group"> <label
for="foo">Supercalifragilistic Label</label> <input id="foo" type="text"
placeholder="Enter something here..."> </div> <div styling="controls"> <label
for="cb" styling="checkbox"> <input id="cb" type="checkbox"> I've read the terms
and conditions </label> <button type="submit" styling="button pure-button-
primary">Submit</button> </div> </fieldset> </form>
```

Multi Column Form (Using Grid)

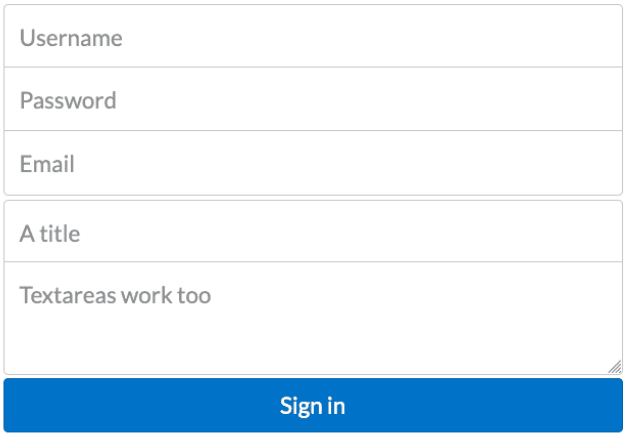
Example



```
<form styling="form form-stacked"> <fieldset> <div class="tk-rgrid-g"> <div
class="tk-rgrid-u-1 tk-rgrid-u-md-1-3"> <label for="first-name">First Name</label>
<input id="first-name" class="tk-rgrid-u-23-24" type="text"> </div> <div class="tk-
rgrid-u-1 tk-rgrid-u-md-1-3"> <label for="last-name">Last Name</label> <input
id="last-name" class="tk-rgrid-u-23-24" type="text"> </div> <div class="tk-rgrid-u-
1 tk-rgrid-u-md-1-3"> <label for="email">E-Mail</label> <input id="email"
class="tk-rgrid-u-23-24" type="email" required=""> </div> <div class="tk-rgrid-u-1
tk-rgrid-u-md-1-3"> <label for="city">City</label> <input id="city" class="tk-
rgrid-u-23-24" type="text"> </div> <div class="tk-rgrid-u-1 tk-rgrid-u-md-1-3">
<label for="state">State</label> <select id="state" styling="input-1-2">
<option>AL</option> <option>CA</option> <option>IL</option> </select> </div> </div>
<label for="terms" styling="checkbox"> <input id="terms" type="checkbox"> I've read
the terms and conditions </label> <button type="submit"
styling="button">Submit</button> </fieldset> </form>
```

Grouped Inputs

Example




```
<form styling="form"> <fieldset styling="group"> <input type="text" styling="input-
1-2" placeholder="Username"> <input type="text" styling="input-1-2"
placeholder="Password"> <input type="email" styling="input-1-2"
placeholder="Email"> </fieldset> <fieldset styling="group"> <input type="text"
styling="input-1-2" placeholder="A title"> <textarea styling="input-1-2"
placeholder="Textareas work too"></textarea> </fieldset> <button type="submit"
styling="button input-1-2">Sign in</button> </form>
```

Input Sizing

Example

.input-1

input-2-3

input-1-2

input-1-3

input-1-4

```
<form styling="form"> <input styling="input-1" type="text" placeholder="input-1">
<br> <input styling="input-2-3" type="text" placeholder="input-2-3"><br> <input
styling="input-1-2" type="text" placeholder="input-1-2"><br> <input styling="input-
1-3" type="text" placeholder="input-1-3"><br> <input styling="input-1-4"
type="text" placeholder="input-1-4"><br> </form>
```

Rounded Inputs

Example

```
<form styling="form"> <input type="text" styling="input-rounded"> </form>
```

Checkboxes and Radio Buttons

Example

Here's option one.

Here's a radio button. You can choose this one..

..Or this one!

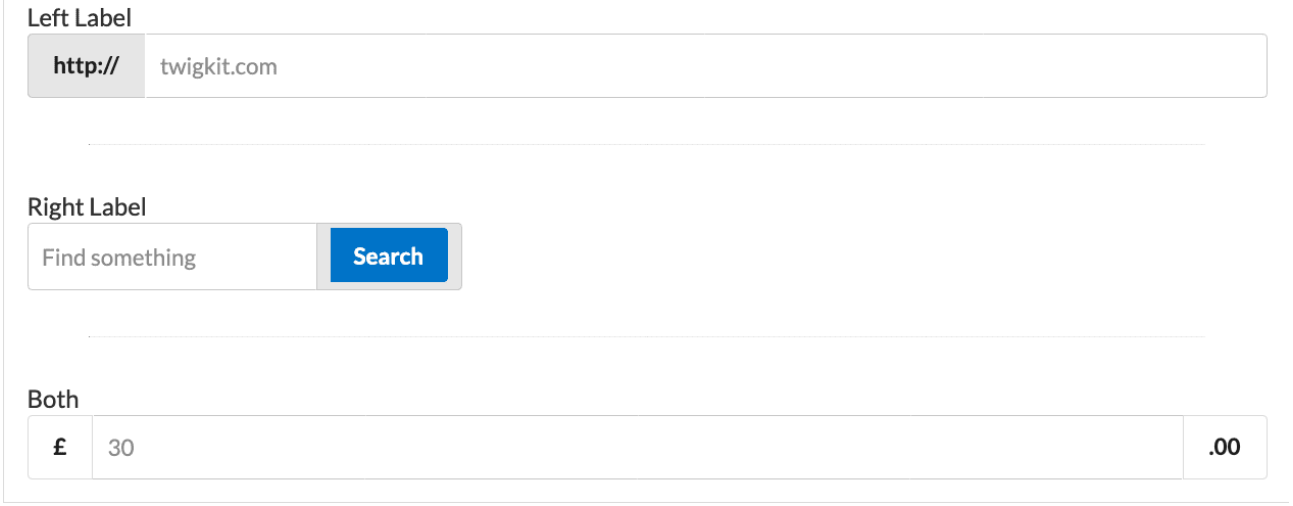
Checkboxes that can be styled

Some label text

```
<form styling="form"> <label for="option-one" styling="checkbox"> <input
id="option-one" type="checkbox" value=""> Here's option one. </label> <label
for="option-two" styling="radio"> <input id="option-two" type="radio"
name="optionsRadios" value="option1" checked=""> Here's a radio button. You can
choose this one.. </label> <label for="option-three" styling="radio"> <input
id="option-three" type="radio" name="optionsRadios" value="option2"> ..Or this one!
</label> <h4>Checkboxes that can be styled</h4> <input id="check1" type="checkbox"
styling="styled-checkbox"> <label for="check1">Some label text</label> </form>
```

Labelled Inputs

Example




```
<p>Left Label</p> <div styling="labelled-input"> <div class="label grey"> <span
class="content">http://</span></div> <input type="text" placeholder="twigkit.com">
</div> <hr> <p>Right Label</p> <div styling="labelled-input label-right input-1-3">
<input type="text" placeholder="Find something"> <div class="label grey"> <button
styling="button">Search</button> </div> </div> <hr> <p>Both</p> <div
styling="labelled-input label-right"> <div class="label basic"> <span
class="content">£</span> </div> <input type="text" placeholder="30"> <div
class="label basic"> <span class="content">.00</span> </div> </div>
```

Grid

Grid System

API	Device width when breakpoint overrides default
default	Sets default layout direction unless overridden by another breakpoint.
@xs-width	width < 35.5 em
@sm-width	600 em ⇐ width < 35.5 em
@md-width	35.5 em ⇐ width < 48 em
@lg-width	48 em ⇐ width < 64 em
@xl-width	width >= 80 em

Twigkit uses a response 24 column grid system. The column widths are specified using a ratio of 24ths.

For example at the attribute `sm="1-3"` to the `layout:block` tag to set the width to 1/3 of the container width on small screens.

By default `layout:blocks` will be 100% width.

Example

```
<layout:grid styling="text-white"> <layout:block xs="1"><layout:box styling="bg-raspberry-dark">1 (Default)</layout:box></layout:block> <layout:block xs="23-24"> <layout:box styling="bg-raspberry">23-24</layout:box></layout:block> <layout:block xs="7-8"><layout:box styling="bg-raspberry-light">7-8</layout:box></layout:block> <layout:block xs="5-6"><layout:box styling="bg-strawberry-light">5-6</layout:box> </layout:block> <layout:block xs="4-5"><layout:box styling="bg-strawberry">4-5</layout:box></layout:block> <layout:block xs="3-4"><layout:box styling="bg-strawberry-dark">3-4</layout:box></layout:block> <layout:block xs="2-3"><layout:box styling="bg-carrot">2-3</layout:box></layout:block> <layout:block xs="1-2"> <layout:box styling="bg-carrot-light">1-2</layout:box></layout:block> </layout:grid>
```

Grid Breakpoints

Widths can be specified for each of the `xs`, `sm`, `md`, `lg`, and `xl` breakpoints individually.

`lg`

Example

`md`

Example

`sm`

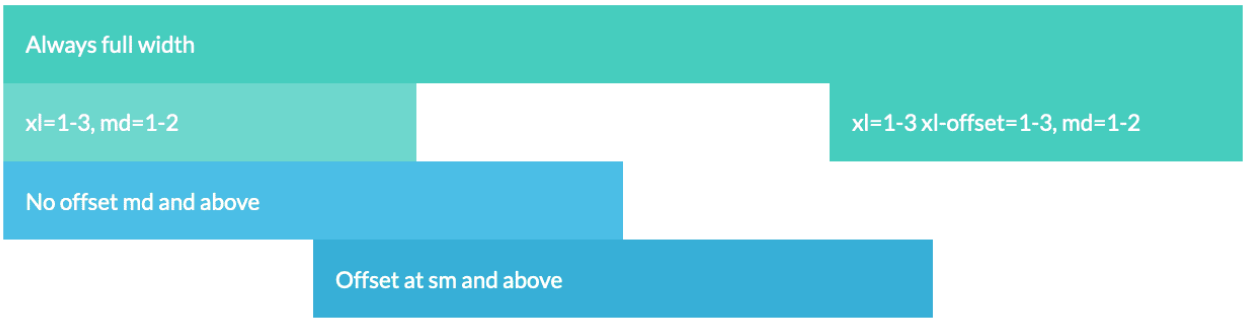
Example

```
<layout:grid styling="text-white"> <layout:block xs="1"> <layout:box styling="bg-sunflower">Always full width</layout:box> </layout:block> <layout:grid> <layout:block xl="1-3" md="1-2"> <layout:box styling="bg-pumpkin-light">xl=1-3, md=1-2</layout:box> </layout:block> <layout:block xl="1-3" md="1-2"> <layout:box styling="bg-pumpkin">xl=1-3, md=1-2</layout:box> </layout:block> <layout:block xl="1-3" md="1"> <layout:box styling="bg-pumpkin-dark">xl=1-3, md=1</layout:box> </layout:block> </layout:grid> </layout:grid>
```

Grid Offsets

An offset attribute can be used to push the `layout:block` to the right, leaving an empty space

Example

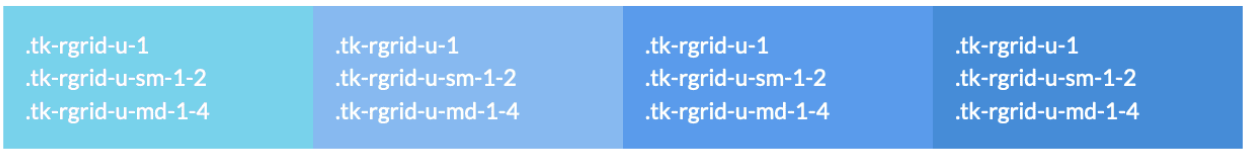



```
<layout:grid styling="text-white"> <layout:block xs="1"> <layout:box styling="bg-mint">Always full width</layout:box> </layout:block> <layout:grid> <layout:block xl="1-3" md="1-2"> <layout:box styling="bg-mint-light">xl=1-3, md=1-2</layout:box> </layout:block> <layout:block xl="1-3" md="1-2" xl-offset="1-3"> <layout:box styling="bg-mint">xl=1-3 xl-offset=1-3, md=1-2</layout:box> </layout:block> </layout:grid> <layout:grid> <layout:block xs="1-2" xs-offset="1-4" sm-offset="1-2" md-offset="0"> <layout:box styling="bg-aqua">No offset md and above</layout:box> </layout:block> </layout:grid> <layout:grid> <layout:block xs="1-2" sm-offset="1-4"> <layout:box styling="bg-aqua-dark">Offset at sm and above</layout:box> </layout:block> </layout:grid> </layout:grid>
```

Nested

Standard Button

Example



```
<layout:grid styling="text-white"> <layout:block xs="1-1" sm="1-2" md="1-4">
<layout:box styling="bg-aqua-light"> .tk-rgrid-u-1 <br> .tk-rgrid-u-sm-1-2 <br>
.tk-rgrid-u-md-1-4 <br> </layout:box> </layout:block> <layout:block xs="1-1" sm="1-
2" md="1-4"> <layout:box styling="bg-sky-light"> .tk-rgrid-u-1 <br> .tk-rgrid-u-sm-
1-2 <br> .tk-rgrid-u-md-1-4 <br> </layout:box> </layout:block> <layout:block xs="1-
1" sm="1-2" md="1-4"> <layout:box styling="bg-sky"> .tk-rgrid-u-1 <br> .tk-rgrid-u-
sm-1-2 <br> .tk-rgrid-u-md-1-4 <br> </layout:box> </layout:block> <layout:block
xs="1-1" sm="1-2" md="1-4"> <layout:box styling="bg-sky-dark"> .tk-rgrid-u-1 <br>
.tk-rgrid-u-sm-1-2 <br> .tk-rgrid-u-md-1-4 <br> </layout:box> </layout:block>
</layout:grid>
```

Image

Standard Button

Example

.tk-rgrid-u-1
.tk-rgrid-u-sm-1-2
.tk-rgrid-u-md-1-4

.tk-rgrid-u-1
.tk-rgrid-u-sm-1-2
.tk-rgrid-u-md-1-4

.tk-rgrid-u-sm-1-2
.tk-rgrid-u-md-2-4

.tk-rgrid-u-1
.tk-rgrid-u-md-1-2

.tk-rgrid-u-1
.tk-rgrid-u-md-1-2

```
<layout:grid> <layout:block xs="1-1" sm="1-2" md="1-4"> <layout:box styling="bg-
paper"> <p> .tk-rgrid-u-1 <br> .tk-rgrid-u-sm-1-2 <br> .tk-rgrid-u-md-1-4 <br> </p>
</layout:box> </layout:block> <layout:block xs="1-1" sm="1-2" md="1-4"> <layout:box
styling="bg-paper-dark"> <p> .tk-rgrid-u-1 <br> .tk-rgrid-u-sm-1-2 <br> .tk-rgrid-
u-md-1-4 <br> </p> </layout:box> </layout:block> <layout:block xs="1-1" sm="1-2"
md="2-4"> <layout:box styling="bg-paper"> <p> .tk-rgrid-u-sm-1-2 <br> .tk-rgrid-u-
md-2-4 <br> </p> <layout:grid> <layout:block sm="1-1" md="1-2"> <layout:box
styling="bg-paper-light"> <p> .tk-rgrid-u-1 <br> .tk-rgrid-u-md-1-2 <br> </p>
</layout:box> </layout:block> <layout:block sm="1-1" md="1-2"> <layout:box
styling="bg-paper-dark"> <p> .tk-rgrid-u-1 <br> .tk-rgrid-u-md-1-2 <br> </p>
</layout:box> </layout:block> </layout:grid> </layout:box> </layout:block>
</layout:grid>
```

Justify Content

Example

Long content

item 1

item2

item 3

```
<layout:grid> <layout:block styling="center-content bg-raspberry text-white" md="1-2"> <p>Long content</p> <br><br><br><br><br><br><br> </layout:block> <layout:block md="1-2"> <div styling="stretch-content"> <layout:block styling="center-content bg-strawberry text-white" md="1-2">item 1</layout:block> <layout:block styling="center-content bg-carrot text-white" md="1-2">item2</layout:block> <layout:block styling="center-content bg-pumpkin text-white full-width">item 3</layout:block> </div> </layout:block> </layout:grid>
```

Hide

You can hide the layout by using hide or show at your desire size point `styling="hide-xs"`

`xl`

Example

`md`

Example

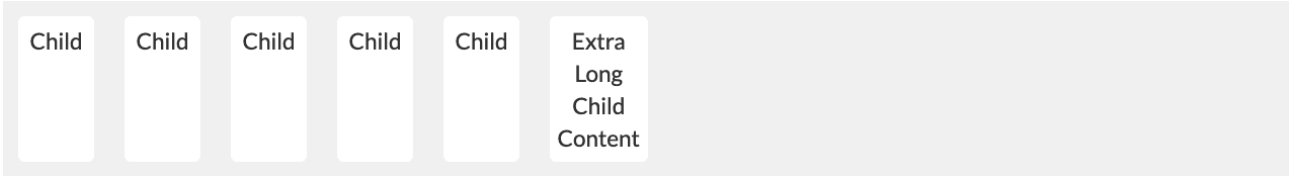
`xs`

Example

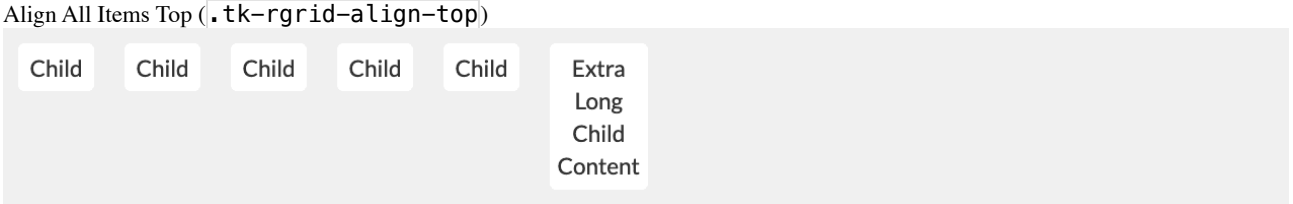
```
<layout:grid> <layout:block md="1-5" styling="hide-xs bg-concrete-light">
<layout:box>hide @xs-width</layout:box> </layout:block> <layout:block md="1-5"
styling="hide-sm bg-concrete"> <layout:box>hide @sm-width</layout:box>
</layout:block> <layout:block md="1-5" styling="hide-md bg-concrete-dark">
<layout:box>hide @md-width</layout:box> </layout:block> <layout:block md="1-5"
styling="hide-lg bg-asphalt-light"> <layout:box>hide @lg-width</layout:box>
</layout:block> <layout:block md="1-5" styling="hide-xl bg-asphalt">
<layout:box>hide @xl-width</layout:box> </layout:block> </layout:grid>
<layout:grid> <layout:block md="1-5" styling="show-xs bg-pink-light">
<layout:box>show @xs-width</layout:box> </layout:block> <layout:block md="1-5"
styling="show-sm bg-pink"> <layout:box>show @sm-width</layout:box> </layout:block>
<layout:block md="1-5" styling="show-md bg-pink-dark"> <layout:box>show @md-
width</layout:box> </layout:block> <layout:block md="1-5" styling="show-lg bg-
lavender"> <layout:box>show @lg-width</layout:box> </layout:block> <layout:block
md="1-5" styling="show-xl bg-lavender-dark"> <layout:box>show @xl-
width</layout:box> </layout:block> </layout:grid>
```

Alignments

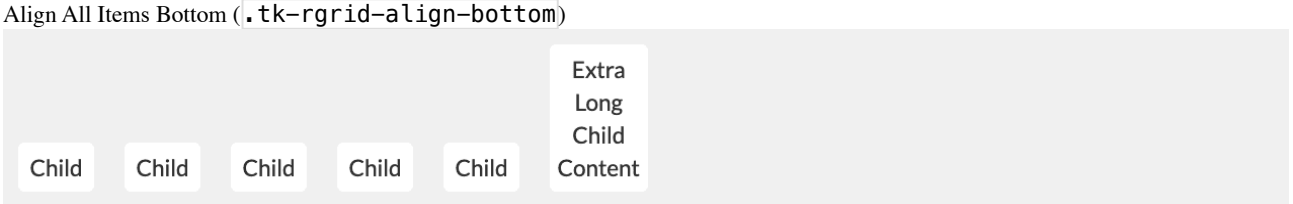
Default (Stretch)



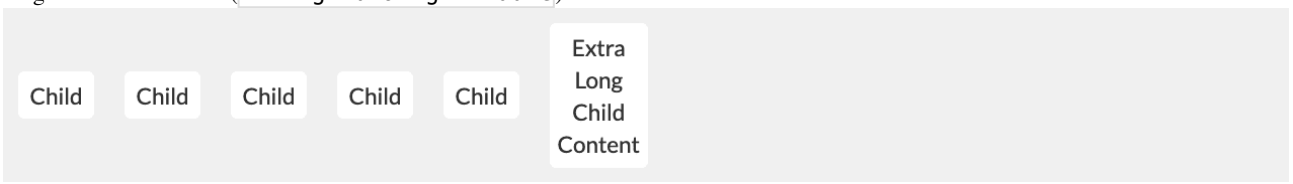
Align All Items Top (`.tk-rgrid-align-top`)



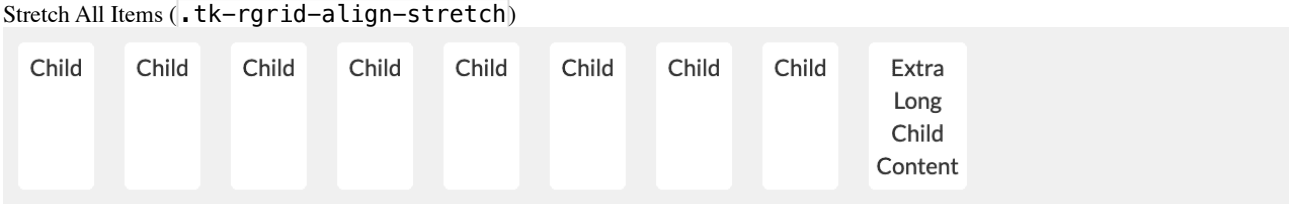
Align All Items Bottom (`.tk-rgrid-align-bottom`)



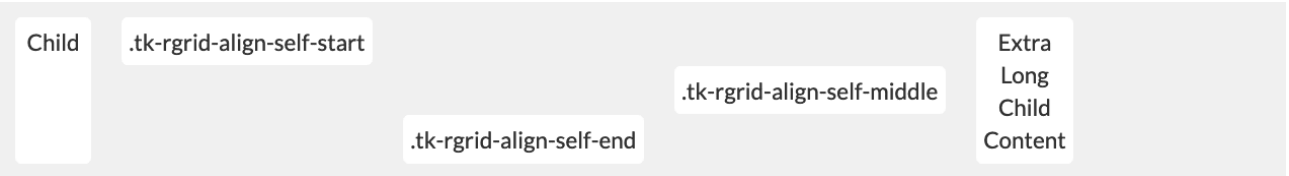
Align All Items Middle (`.tk-rgrid-align-middle`)



Stretch All Items (`.tk-rgrid-align-stretch`)

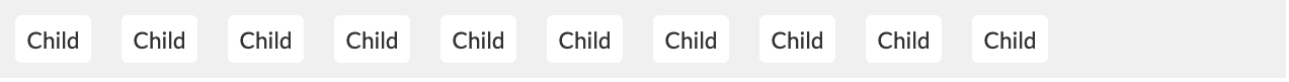


Item Alignment Overrides

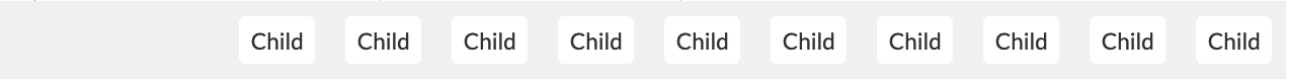


Justify Items

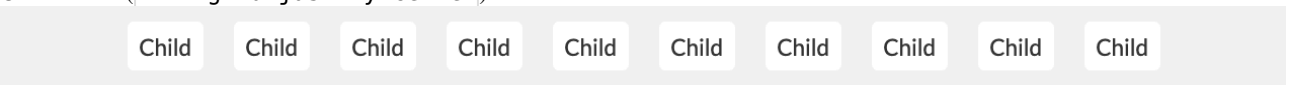
Default, Align items to the start (`.tk-rgrid-justify-start`)



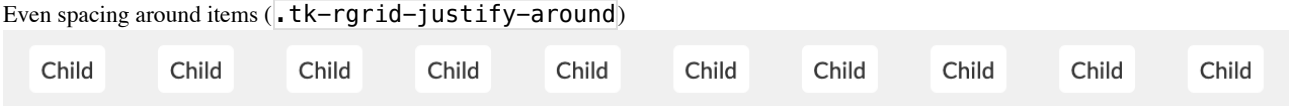
Align items to the end of the container (`.tk-rgrid-justify-end`)



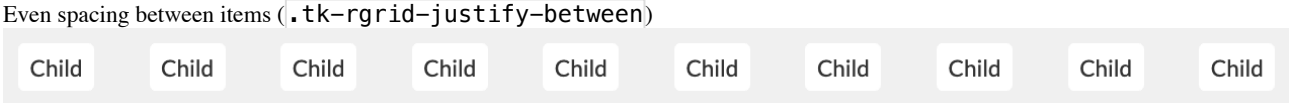
Center items (`.tk-rgrid-justify-center`)



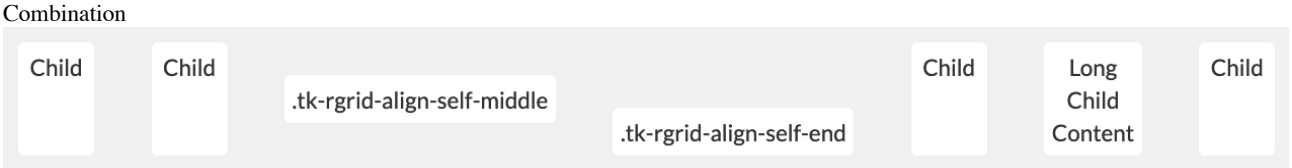
Even spacing around items (`.tk-rgrid-justify-around`)



Even spacing between items (`.tk-rgrid-justify-between`)

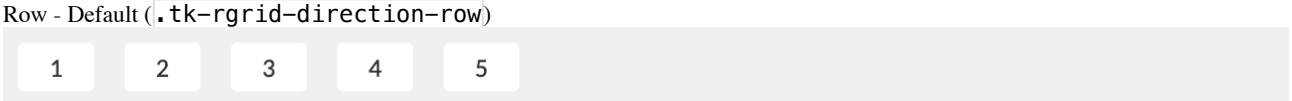


Combination

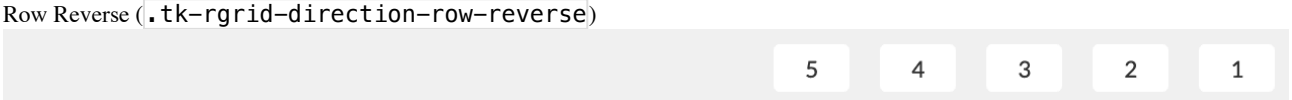


Container Direction

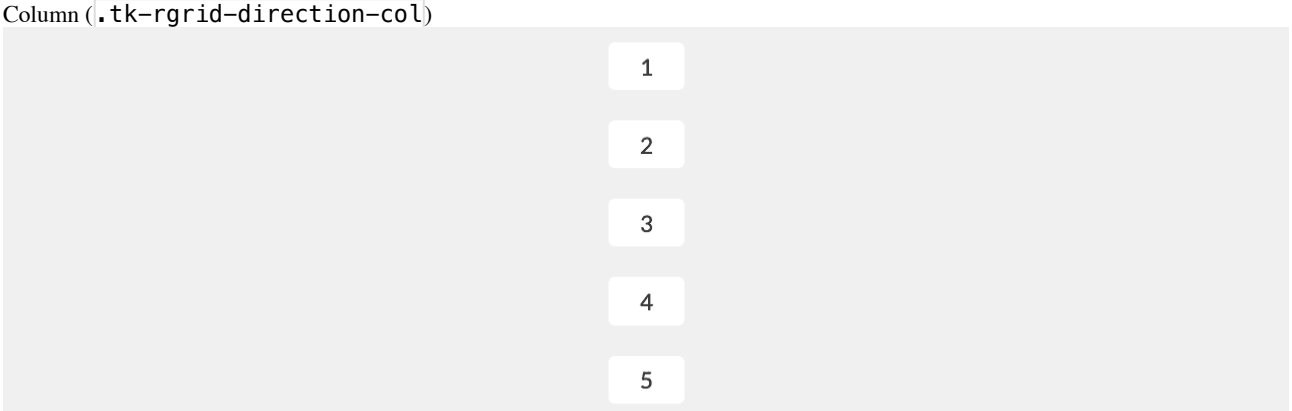
Row - Default (`.tk-rgrid-direction-row`)



Row Reverse (`.tk-rgrid-direction-row-reverse`)



Column (`.tk-rgrid-direction-col`)



Column Reverse (`.tk-rgrid-direction-col-reverse`)

5

4

3

2

1

Offsets

.tk-rgrid-u-1
.tk-rgrid-u-sm-1-2
.tk-rgrid-u-md-1-4
.tk-rgrid-u-offset-sm-1-2
.tk-rgrid-u-offset-md-1-4

.tk-rgrid-u-1
.tk-rgrid-u-sm-1-2
.tk-rgrid-u-md-1-4
.tk-rgrid-u-offset-sm-1-2
.tk-rgrid-u-offset-md-1-4

Icons

Twigkit provide a handful of useful icons for you to use.

icon-doc
icon-doc-text
icon-doc-inv
icon-doc-text-inv

icon-file-pdf
icon-file-word
icon-file-excel
icon-file-powerpoint
icon-file-image
icon-file-archive
icon-file-audio
icon-file-video
icon-file-code

---

icon-folder-empty
icon-folder-open-empty
icon-folder
icon-folder-open

icon-user-o
icon-user
icon-user-circle
icon-user-circle-o
icon-user-times
icon-user-plus
icon-address-book-o
icon-address-book
icon-address-card-o
icon-address-card
icon-id-card-o

---

icon-twitter
icon-facebook
icon-linkedin
icon-youtube
icon-google
icon-skype
icon-slack
icon-whatsapp
icon-stackoverflow
icon-twitter-squared
icon-facebook-squared
icon-linkedin-squared
icon-youtube-squared
icon-gplus-squared

---

icon-heart
icon-heart-empty
icon-star-empty
icon-star-half-alt
icon-star
icon-bookmark-empty
icon-bookmark
icon-thumbs-up
icon-thumbs-down
icon-thumbs-up-alt
icon-thumbs-down-alt
icon-flag-empty
icon-flag
icon-flag-checkered
icon-flag-1

icon-tag
icon-tags

---

icon-location
icon-pin
icon-lightbulb
icon-direction

---

icon-comment
icon-chat
icon-chat-empty

---

icon-resize-full
icon-resize-small
icon-television
icon-move
icon-resize-full-alt
icon-zoom-in
icon-zoom-out
icon-window-maximize
icon-window-minimize

---

icon-check
icon-check-empty

---

icon-chart-bar
icon-chart-pie
icon-chart-area
icon-chart-line
icon-chart-line-1

---

icon-cog
icon-share
icon-link
icon-link-ext
icon-attach
icon-code
icon-export
icon-upload
icon-download
icon-upload-cloud
icon-download-cloud
icon-clone
icon-lock
icon-lock-open-alt
icon-trash
icon-trash-empty

---

icon-info-circled
icon-help-circled
icon-ok-circled
icon-plus-circled
icon-minus-circled
icon-cancel-circled
icon-info
icon-help
icon-ok
icon-plus
icon-minus
icon-cancel
icon-attention

---

icon-home
icon-search
icon-filter
icon-sliders
icon-signal
icon-ellipsis-vert
icon-ellipsis
icon-sort-name-up
icon-sort-name-down
icon-sort-number-up
icon-sort-number-down
icon-sort-alt-up
icon-sort-alt-down
icon-table
icon-sitemap
icon-menu
icon-th-large
icon-th
icon-th-list
icon-align-justify

---

icon-angle-up
icon-angle-down
icon-angle-left
icon-angle-right
icon-up-dir
icon-down-dir
icon-left-dir
icon-right-dir
icon-up-open-1
icon-down-open-1
icon-left-open-1
icon-right-open-1
icon-up-open
icon-down-open
icon-left-open
icon-right-open

---

icon-up-big
icon-down-big
icon-left-big
icon-right-big

---

icon-bell
icon-bell-alt
icon-bell-off-empty
icon-bell-off

---

icon-calendar
icon-clock
icon-clock-1
icon-history

---

icon-mail
icon-inbox

---

icon-shopping-basket
icon-box
icon-sticky-note
icon-sticky-note-o

---

icon-stethoscope
icon-beaker
icon-thermometer
icon-thermometer-3
icon-thermometer-2
icon-thermometer-

quarter
icon-thermometer-0

Input

Standard

Example

Search...

```
<div styling="input"> <input type="text" placeholder="Search..."> </div>
```

Disabled

Example

Search...

```
<input type="text" placeholder="Search..." disabled="">
```

Icon

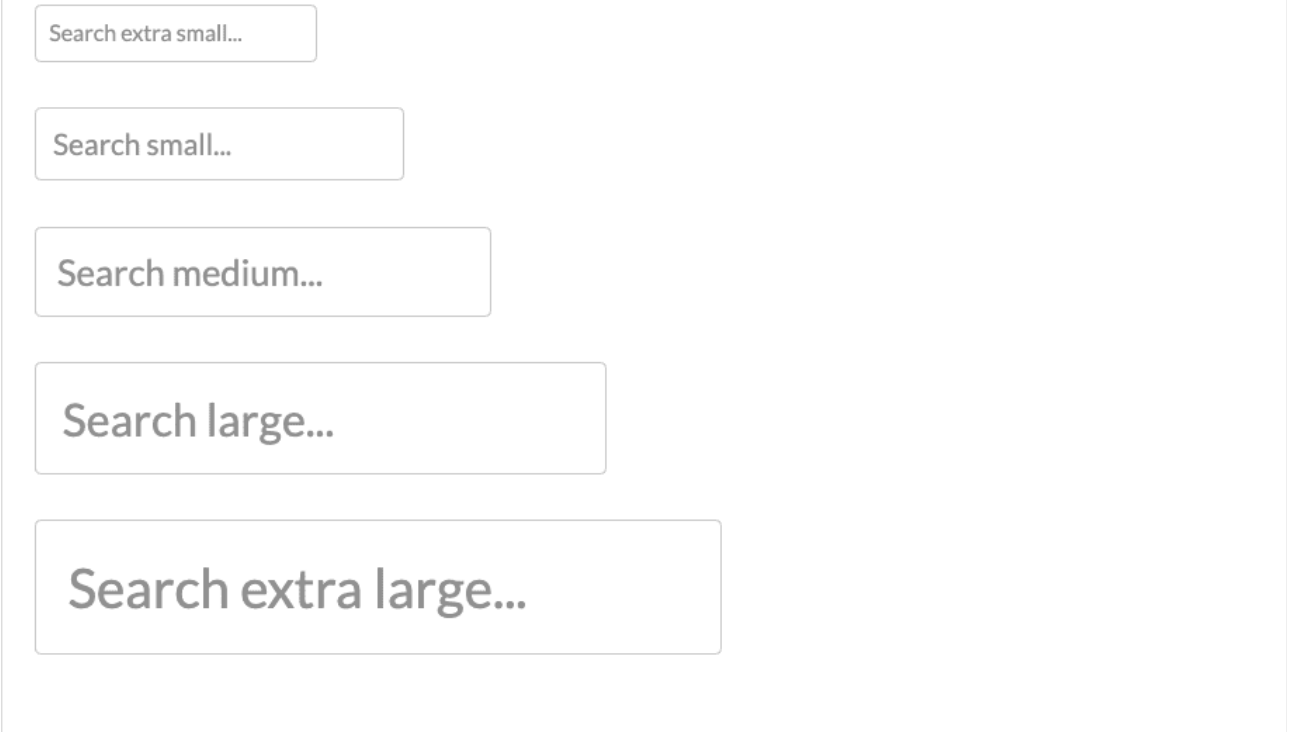
Example

Search...

```
<div styling="icon input"> <input type="text" placeholder="Search..."> <i
class="icon-search3 icon"></i> </div>
```

Size

Example




```
<input type="text" placeholder="Search extra small..." styling="xs"> <input
type="text" placeholder="Search small..." styling="sm"> <input type="text"
placeholder="Search medium..." styling="md"> <input type="text" placeholder="Search
large..." styling="lg"> <input type="text" placeholder="Search extra large..."
styling="xl">
```

Loader

Rotating spinner

Example

```
<widget:spinner></widget:spinner>
```

Message

Interruptor

Example

I'm an Interruptor

Interruptors are smaller than banners, but larger than alertboxes.
They're about drawing your attention to a CTA of some sort.

Button

```
<div styling="interruptor"> <h4>I'm an Interruptor</h4> <p>Interruptors are smaller than banners, but larger than alertboxes.<br> They're about drawing your attention to a CTA of some sort.</p> <p><a styling="button small disabled">Button</a></p> </div>
```

Alert Box

Example

Did you mean the shop kitchensinks.co.uk?

Something you can probably ignore has happened.

Something bad has happened.

Something slightly worrying has happened.

Something good has happened.

```
<div styling="alertbox"> Did you mean the shop <a href="">kitchensinks.co.uk</a>?
<a href="" class="close">x</a> </div> <div styling="alertbox alertbox-secondary">
Something you can probably ignore has happened. <a href="" class="close">x</a>
</div> <div styling="alertbox alertbox-negative"> Something bad has happened. <a
href="" class="close">x</a> </div> <div styling="alertbox alertbox-warning">
Something slightly worrying has happened. <a href="" class="close">x</a> </div>
<div styling="alertbox alertbox-positive"> Something good has happened. <a href=""
class="close">x</a> </div>
```

Pagination

Default

Example

1 2 3 4 5 next

```
<search:pagination response="response" max-pages="5"></search:pagination>
```

Centered

Example

1 2 3 4 5 next

```
<search:pagination response="response" max-pages="5" styling="pagination-wrapper-
center"></search:pagination>
```

Right

Example

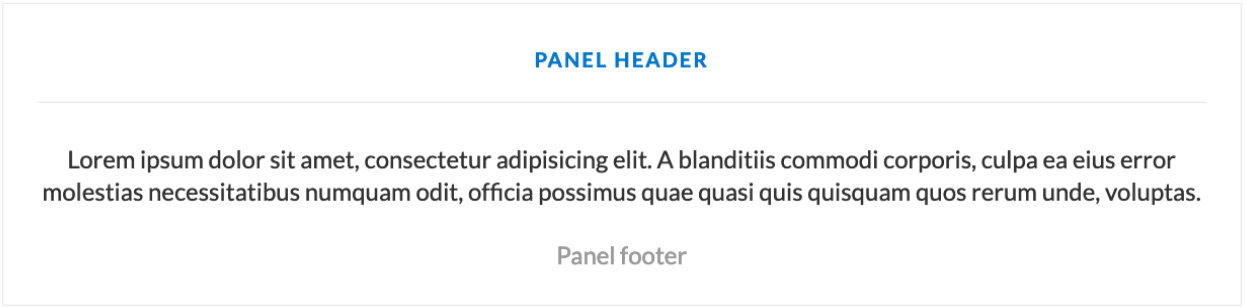
1 2 3 4 5 next

```
<search:pagination response="response" max-pages="5" styling="pagination-wrapper-
right"></search:pagination>
```

Panel

Default

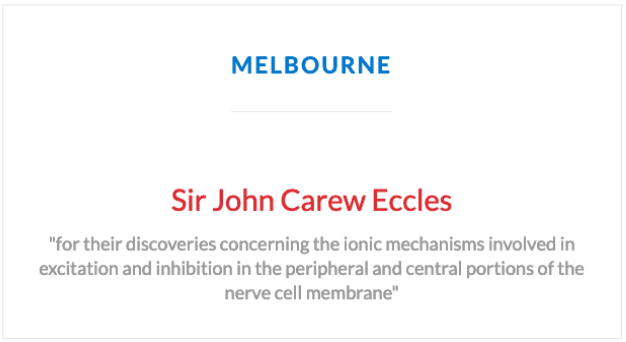
Example




```
<div styling="panel"> <h3 styling="title">Panel Header</h3> <p>Lorem ipsum dolor sit amet, consectetur adipisicing elit. A blanditiis commodi corporis, culpa ea eius error molestias necessitatibus numquam odit, officia possimus quae quasi quis quisquam quos rerum unde, voluptas.</p> <br> <div styling="footer"> <p>Panel footer</p> </div> </div>
```

Using a result

Example




```
<search:result result="response | result:5"> <layout:grid> <layout:block md="1-2"> <div styling="stretch-content"> <div styling="center aligned panel center-content"> <search:field name="born_city" styling="title"></search:field> <search:field name="full_name" styling="description" value-element="h3"></search:field> <div styling="footer"> <search:field name="motivation" styling="description"> </search:field> </div> </div> </div> </layout:block> </layout:grid> </search:result>
```

Progress Bars

Default

Example

```
<div styling="progress-bar"> <div styling="progress-bar-fill" style="width: 66%">
</div> </div>
```

Background Color

All background colors can be use to change the color of the bar

Example


```
<div styling="progress-bar"> <div styling="progress-bar-fill bg-sunflower"
style="width: 33%"></div> </div>
```

Radial

Example

```
<layout:grid> <layout:block sm="1-2"> <progress:radial percentage="33">
</progress:radial> </layout:block> <layout:block sm="1-2"> <progress:radial
percentage="88" fill-class="tk-stl-bg-raspberry" percentage-class="tk-stl-text-
raspberry"></progress:radial> </layout:block> </layout:grid>
```

Result List

Default

The default result list shows all the fields

Example

id	937
firstname	Bob
surname	Dylan
born	1941-05-24T00:00:00
born_lat	46.8
born_lon	-92.1
born_coords	46.8,-92.1
Country of birth	USA
born_countrycode	US
born_city	Duluth, MN
gender	Male
facialhair	moustache
year	2016
Prize Category	literature
share	1
motivation	"for having created new poetic expressions within the great American song tradition"
age	75
	75
type	laureate
award_id	937-2016-literature
prize_id	2016-literature
version	1548651151257239600
score	1
full_name	Bob Dylan
photo_url	assets/avatars/a.jpg
photo	assets/img2.jpg
details_url	/#/laureates/937
affiliation_address	,

```
<search:result-list response="smallResponse"></search:result-list>
```

List

Full width list of results

Example

Bob Dylan

Year 2016 Gender Male Prize Category literature

"for having created new poetic expressions within the great American song tradition"

http://nobel.divedeeper.io/#/laureates/937

```
<search:result-list response="smallResponse"> <search:result> <layout:grid>
<layout:block md="1-4" padding-right="1"> <img src="{{ result | field:'photo' |
actual }}" alt=""> </layout:block> <layout:block md="3-4"> <search:field
name="full_name" styling="title"></search:field> <search:field name="year"
label="Year" styling="label-inline"></search:field> <search:field name="gender"
label="Gender" styling="label-inline"></search:field> <search:field name="Prize
Category" styling="label-inline"></search:field> <br> <search:field
name="motivation" styling="description"></search:field> <search:field
name="details_url" styling="url description" prefix="http://nobel.divedeeper.io"
urlfield="details_url" url-prefix="http://nobel.divedeeper.io"></search:field>
</layout:block> </layout:grid> </search:result> </search:result-list>
```

Cards

Result cards that can be shown in a row

Example

Bob Dylan

Year 2016 Gender Male

Prize Category literature

"for having created new poetic expressions within the great American song tradition"

http://nobel.divedeeper.io/#/laureates/937

Karl Ferdinand Braun

Year 1909 Gender Male

Prize Category physics

"in recognition of their contributions to the development of wireless telegraphy"

http://nobel.divedeeper.io/#/laureates/14

Luis F. Leloir

Year 1970 Gender Male

Prize Category chemistry

"for his discovery of sugar nucleotides and their role in the biosynthesis of carbohydrates"

http://nobel.divedeeper.io/#/laureates/239

Bernardo Alberto Houssay

Year 1947 Gender Male

Prize Category medicine

"for his discovery of the part played by the hormone of the anterior pituitary lobe in the metabolism of sugar"

http://nobel.divedeeper.io/#/laureates/345

Sir Frank Macfarlane Burnet

Year 1960 Gender Male

Prize Category medicine

"for discovery of acquired immunological tolerance"

http://nobel.divedeeper.io/#/laureates/369

Sir John Carew Eccles

Year 1963 Gender Male

Prize Category medicine

"for their discoveries concerning the ionic mechanisms involved in excitation and inhibition in the peripheral and central portions of the nerve cell membrane"

http://nobel.divedeeper.io/#/laureates/375

Barry J. Marshall

Year 2005 Gender Male

Prize Category medicine

"for their discovery of the bacterium *Helicobacter pylori* and its role in gastritis and peptic ulcer disease"

http://nobel.divedeeper.io/#/laureates/789

J. Robin Warren

Year 2005 Gender Male

Prize Category medicine

"for their discovery of the bacterium *Helicobacter pylori* and its role in gastritis and peptic ulcer disease"

http://nobel.divedeeper.io/#/laureates/790

Brian P. Schmidt

Year 2011 Gender Male

Prize Category physics

"for the discovery of the accelerating expansion of the Universe through observations of distant supernovae"

http://nobel.divedeeper.io/#/laureates/865

Victor Franz Hess

Year 1936 Gender Male

Prize Category physics

"for his discovery of cosmic radiation"

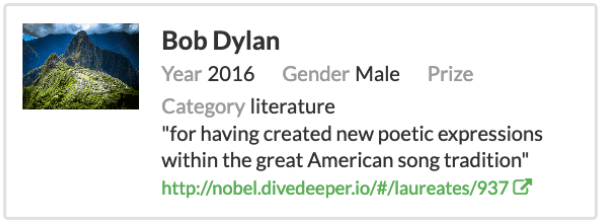
http://nobel.divedeeper.io/#/laureates/42

```
<search:result-list response="response" styling="cards-md-2 cards-xl-3">
<search:result> <layout:grid> <layout:block md="1-4" padding-right="1"> <img src="
{{ result | field:'photo' | actual }}" alt=""> </layout:block> <layout:block md="3-
4"> <search:field name="full_name" styling="title"></search:field> <search:field
name="year" label="Year" styling="label-inline"></search:field> <search:field
name="gender" label="Gender" styling="label-inline"></search:field> <search:field
name="Prize Category" styling="label-inline"></search:field> <br> <search:field
name="motivation" styling="description"></search:field> <search:field
name="details_url" styling="url description" prefix="http://nobel.divedeeper.io"
urlfield="details_url" url-prefix="http://nobel.divedeeper.io" target="_blank">
</search:field> </layout:block> </layout:grid> </search:result> </search:result-
list>
```

Card Emphasis

A result card with a little more border

Example




```
<search:result-list response="smallResponse" styling="cards-md-2"> <search:result
styling="card-emphasis"> <layout:grid> <layout:block md="1-4" padding-right="1">
<img src="{{ result | field:'photo' | actual }}" alt=""> </layout:block>
<layout:block md="3-4"> <search:field name="full_name" styling="title">
</search:field> <search:field name="year" label="Year" styling="label-inline">
</search:field> <search:field name="gender" label="Gender" styling="label-inline">
</search:field> <search:field name="Prize Category" styling="label-inline">
</search:field> <br> <search:field name="motivation" styling="description">
</search:field> <search:field name="details_url" styling="url description"
prefix="http://nobel.divedeeper.io" urlfield="details_url" url-
prefix="http://nobel.divedeeper.io" target="_blank"></search:field> </layout:block>
</layout:grid> </search:result> </search:result-list>
```

Card Shadow

Add a small drop shadow

Example

Bob Dylan

Year 2016 Gender Male Prize Category literature

"for having created new poetic expressions within the great American song tradition"

http://nobel.divedeeper.io/#/laureates/937

```
<search:result-list response="smallResponse" styling="cards-md-2"> <search:result styling="card-shadow"> <layout:grid> <layout:block md="1-4" padding-right="1"> <img src="{{ result | field:'photo' | actual }}" alt=""> </layout:block> <layout:block md="3-4"> <search:field name="full_name" styling="title"></search:field> <search:field name="year" label="Year" styling="label-inline"></search:field> <search:field name="gender" label="Gender" styling="label-inline"></search:field> <search:field name="Prize Category" styling="label-inline"></search:field> <br> <search:field name="motivation" styling="description"></search:field> <search:field name="details_url" styling="url description" prefix="http://nobel.divedeeper.io" urlfield="details_url" url-prefix="http://nobel.divedeeper.io" target="_blank"> </search:field> </layout:block> </layout:grid> </search:result> </search:result-list>
```

Card Step

A result card with a slightly raised edge

Example

Bob Dylan

Year 2016 Gender Male Prize Category literature

"for having created new poetic expressions within the great American song tradition"

http://nobel.divedeeper.io/#/laureates/937

```
<search:result-list response="smallResponse" styling="cards-md-2"> <search:result styling="card-step"> <layout:grid> <layout:block md="1-4" padding-right="1"> <img src="{{ result | field:'photo' | actual }}" alt=""> </layout:block> <layout:block md="3-4"> <search:field name="full_name" styling="title"></search:field> <search:field name="year" label="Year" styling="label-inline"></search:field> <search:field name="gender" label="Gender" styling="label-inline"></search:field> <search:field name="Prize Category" styling="label-inline"></search:field> <br> <search:field name="motivation" styling="description"></search:field> <search:field name="details_url" styling="url description" prefix="http://nobel.divedeeper.io" urlfield="details_url" url-prefix="http://nobel.divedeeper.io" target="_blank"> </search:field> </layout:block> </layout:grid> </search:result> </search:result-list>
```

Table

Comparison Tables

Different options for displaying tables on small screens. In terms of CSS, examples 1 and 2 are identical. Example 3 stacks, so is perhaps unsuitable for comparisons. Example 4 is more of a hybrid that is in line with our original sketches. Examples three and four both rely on `data-title` for the labels.

Tested in latest Chrome, FireFox, Safari and Opera. IE10 and later works. IE9 should be possible with some effort.

Tables

Vertical Comparison

	Henson Pictures	Sony Pictures	Warner Bros.	Walt Disney Studios
URL	henson.com	sonypictures.com	warnerbros.com	disney.com
CEO	Brian Henson	Thomas Rothman	Kevin Tsujihara	Alan Horn
Headquarters	Los Angeles, California	Culver City, California	Burbank, California	Burbank, California
Founded	November 20, 1958	August 7, 1991	April 4, 1923	November 05, 1936

Horizontal Comparison

	Henson Pictures	Sony Pictures	Warner Bros.	Walt Disney Studios
URL	henson.com	sonypictures.com	warnerbros.com	disney.com
CEO	Brian Henson	Thomas Rothman	Kevin Tsujihara	Alan Horn
Headquarters	Los Angeles, California	Culver City, California	Burbank, California	Burbank, California
Founded	November 20, 1958	August 7, 1991	April 4, 1923	November 05, 1936

Stacked

Name	URL	CEO	Headquarters	Founded
Henson Pictures	henson.com	Brian Henson	Los Angeles, California	November 20, 1958
Sony Pictures	sonypictures.com	Thomas Rothman	Culver City, California, United States	August 7, 1991
Warner Bros.	warnerbros.com	Kevin Tsujihara	Burbank, California	April 4, 1923
Walt Disney Studios	disney.com	Alan Horn	Burbank, California	November 05, 1936
Universal Studios	universalstudios.com	Ronald Meyer	Universal City, California	April 10, 1912
United Artists	uamg.com	Mark Burnett	Beverly Hills, California	February 5, 1919
Metro-Goldwyn-Mayer	mgm.com	Gary Barber	Los Angeles, California	April 17, 1924

Stacked Comparison

Name	URL	CEO	Headquarters	Founded
Henson Pictures	henson.com	Brian Henson	Los Angeles, California	November 20, 1958
Sony Pictures	sonypictures.com	Thomas Rothman	Culver City, California, United States	August 7, 1991
Warner Bros.	warnerbros.com	Kevin Tsujihara	Burbank, California	April 4, 1923
Walt Disney Studios	disney.com	Alan Horn	Burbank, California	November 05, 1936
Universal Studios	universalstudios.com	Ronald Meyer	Universal City, California	April 10, 1912
United Artists	uamg.com	Mark Burnett	Beverly Hills, California	February 5, 1919
Metro-Goldwyn-Mayer	mgm.com	Gary Barber	Los Angeles, California	April 17, 1924

Combination Comparison

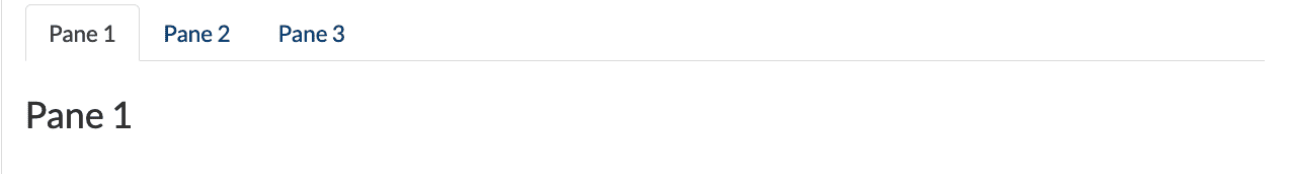
	Henson Pictures	Sony Pictures	Warner Bros.	Walt Disney Studios
URL	henson.com	sonypictures.com	warnerbros.com	disney.com
CEO	Brian Henson	Thomas Rothman	Kevin Tsujihara	Alan Horn
Headquarters	Los Angeles, California	Culver City, California	Burbank, California	Burbank, California
Founded	November 20, 1958	August 7, 1991	April 4, 1923	November 05, 1936

Tabs and Toggle Panes

Default

The standard toggle pane tabs

Example

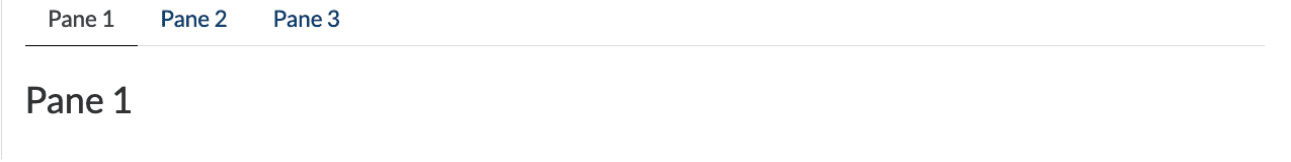



```
<toggle:controls labels="Pane 1, Pane 2, Pane 3" panes="panes"></toggle:controls>
<toggle:panes id="panes"> <toggle:pane> <h2>Pane 1</h2> </toggle:pane>
<toggle:pane> <h2>Pane 2</h2> </toggle:pane> <toggle:pane> <h2>Pane 3</h2>
</toggle:pane> </toggle:panes>
```

Line

Underlined tabs

Example




```
<toggle:controls labels="Pane 1, Pane 2, Pane 3" panes="line" styling="line">
</toggle:controls> <toggle:panes id="line"> <toggle:pane> <h2>Pane 1</h2>
</toggle:pane> <toggle:pane> <h2>Pane 2</h2> </toggle:pane> <toggle:pane> <h2>Pane
3</h2> </toggle:pane> </toggle:panes>
```

Pills

Rounded tabs

Example

```
<toggle:controls labels="Pane 1, Pane 2, Pane 3" panes="pills" styling="pill">
</toggle:controls> <toggle:panes id="pills"> <toggle:pane visible="true"> <h2>Pane
1</h2> </toggle:pane> <toggle:pane> <h2>Pane 2</h2> </toggle:pane> <toggle:pane>
<h2>Pane 3</h2> </toggle:pane> </toggle:panes>
```

Simple

No styling

Example

Pane 1 Pane 2 Pane 3

Pane 1

```
<toggle:controls labels="Pane 1, Pane 2, Pane 3" panes="simple" styling="simple">
</toggle:controls> <toggle:panes id="simple"> <toggle:pane visible="true"> <h2>Pane
1</h2> </toggle:pane> <toggle:pane> <h2>Pane 2</h2> </toggle:pane> <toggle:pane>
<h2>Pane 3</h2> </toggle:pane> </toggle:panes>
```

Bordered

Example

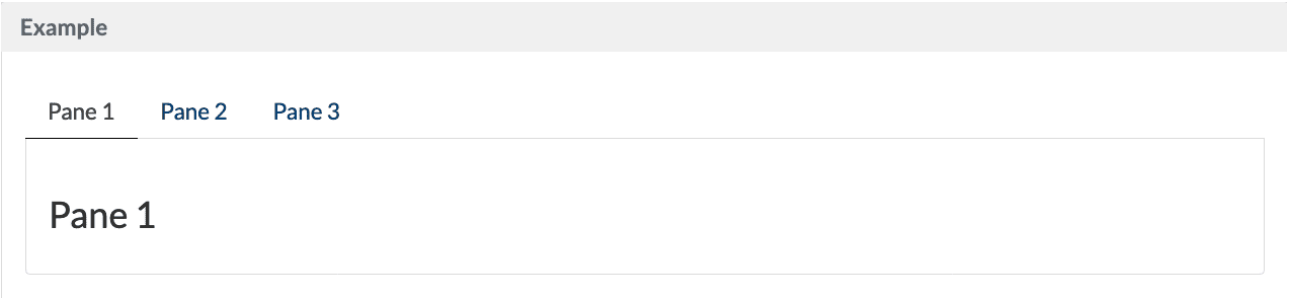
Pane 1 Pane 2 Pane 3

Pane 1

```
<toggle:panes id="bordered"> <toggle:pane visible="true"> <h2>Pane 1</h2>
</toggle:pane> <toggle:pane> <h2>Pane 2</h2> </toggle:pane> <toggle:pane> <h2>Pane
3</h2> </toggle:pane> </toggle:panes>
```

Line Bordered

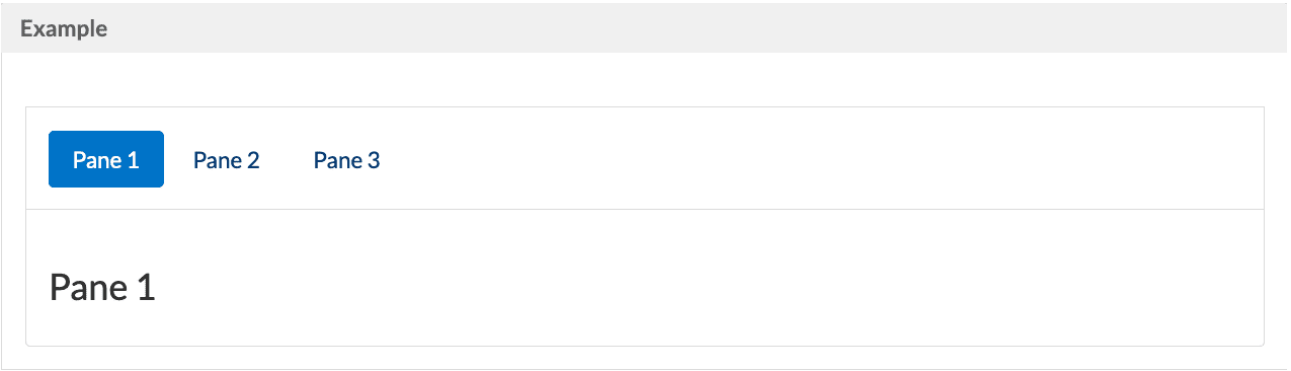
Example




```
<toggle:controls labels="Pane 1, Pane 2, Pane 3" panes="line-b" styling="line bordered"></toggle:controls> <toggle:panes id="line-b"> <toggle:pane visible="true"> <h2>Pane 1</h2> </toggle:pane> <toggle:pane> <h2>Pane 2</h2> </toggle:pane> <toggle:pane> <h2>Pane 3</h2> </toggle:pane> </toggle:panes>
```

Pills Bordered

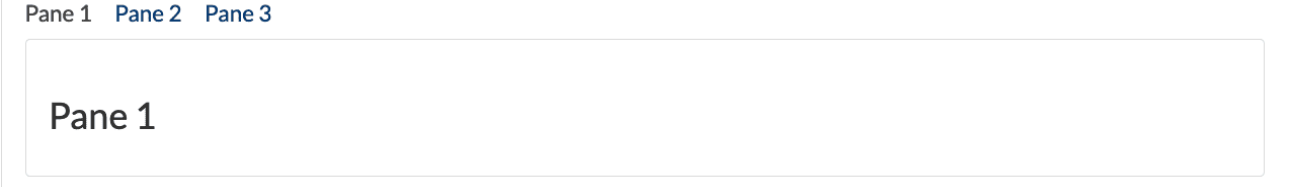
Example




```
<toggle:controls labels="Pane 1, Pane 2, Pane 3" panes="pills-b" styling="pill bordered"></toggle:controls> <toggle:panes id="pills-b"> <toggle:pane visible="true"> <h2>Pane 1</h2> </toggle:pane> <toggle:pane> <h2>Pane 2</h2> </toggle:pane> <toggle:pane> <h2>Pane 3</h2> </toggle:pane> </toggle:panes>
```

Simple Bordered

Example



```
<toggle:controls labels="Pane 1, Pane 2, Pane 3" panes="simple-b" styling="simple bordered"></toggle:controls> <toggle:panes id="simple-b" visible="true"> <toggle:pane visible="true"> <h2>Pane 1</h2> </toggle:pane> <toggle:pane> <h2>Pane 2</h2> </toggle:pane> <toggle:pane> <h2>Pane 3</h2> </toggle:pane> </toggle:panes>
```

Tag

Default / Flat

Example

```
<ol styling="tag-list tag-capsules"> <li> <a>Police</a> <span class="icon-cross2 icon"></span> </li> <li> <a>Martial Arts</a> </li> <li> <a>Comedy</a> </li> <li> <a>Corruption</a> </li> <li> <a>Action</a> </li> </ol>
```

Gradient

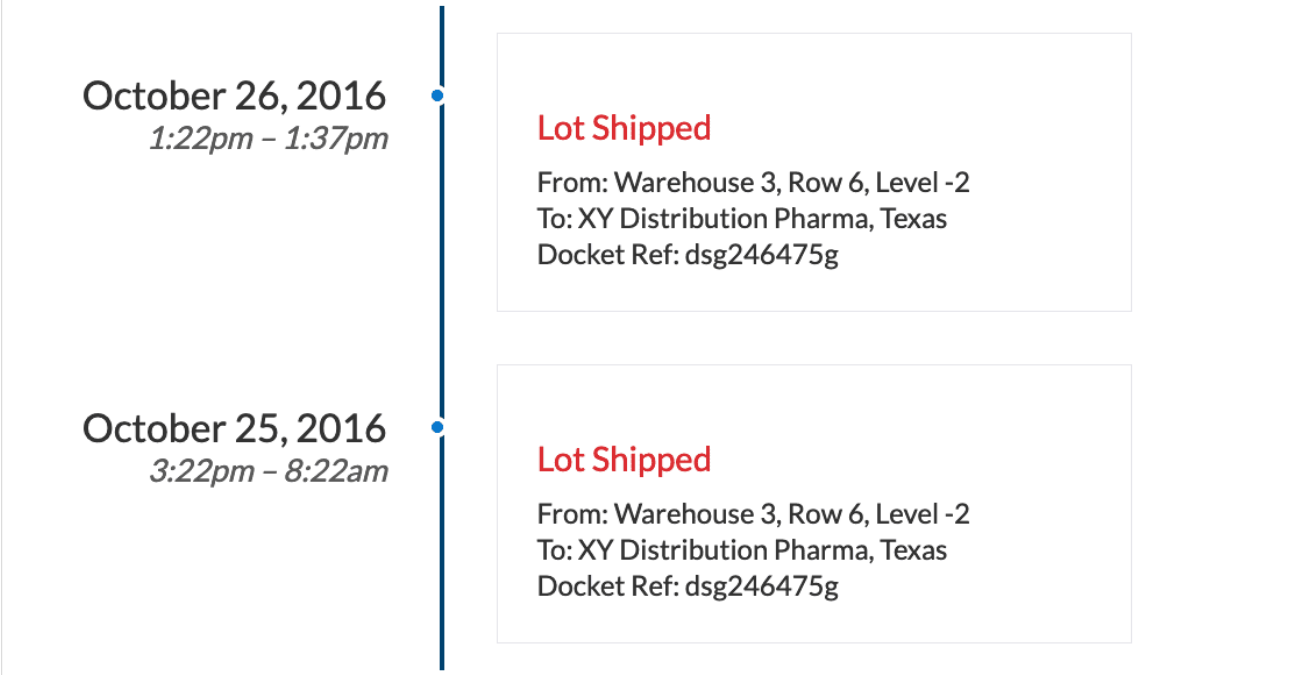
Example

```
<ol styling="tag-list tag-capsules tag-capsules-gradient"> <li> <a>Police</a> </li> <li> <a>Martial Arts</a> </li> <li> <a>Comedy</a> </li> <li> <a>Corruption</a> </li> <li> <a>Action</a> </li> </ol>
```

Timeline

Default

Example



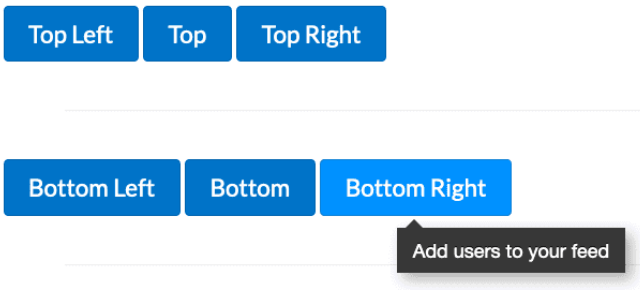

```
<layout:grid> <layout:block> <!-- Entry --> <div styling="timeline-entry">
<layout:grid> <layout:block medium="7-24" large="1-3" styling="timeline-time">
<layout:box padding-right="2"> <h2>October 26, 2016</h2> <h4>1:22pm – 1:37pm</h4>
</layout:box> </layout:block> <layout:block medium="17-24" large="14-24"
styling="timeline-content"> <layout:box padding-left="2"> <div styling="panel text-
left"> <h3>Lot Shipped</h3> <p>From: Warehouse 3, Row 6, Level -2</p> <p>To: XY
Distribution Pharma, Texas</p> <p>Docket Ref: dsg246475g</p> </div> </layout:box>
</layout:block> </layout:grid> </div> <!-- Entry --> <div styling="timeline-entry">
<layout:grid> <layout:block medium="7-24" large="1-3" styling="timeline-time">
<layout:box padding-right="2"> <h2>October 25, 2016</h2> <h4>3:22pm – 8:22am</h4>
</layout:box> </layout:block> <layout:block medium="17-24" large="14-24"
styling="timeline-content"> <layout:box padding-left="2"> <div styling="panel text-
left"> <h3>Lot Shipped</h3> <p>From: Warehouse 3, Row 6, Level -2</p> <p>To: XY
Distribution Pharma, Texas</p> <p>Docket Ref: dsg246475g</p> </div> </layout:box>
</layout:block> </layout:grid> </div> </layout:block> </layout:grid>
```

Tooltip

Positions

Tooltip position variations

Example

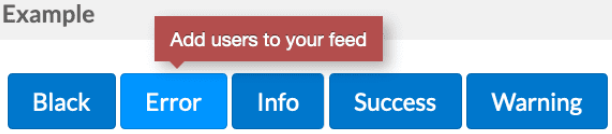



```
<button tooltip="Add users to your feed" styling="tooltip-top-left button"> Top Left </button> <button tooltip="Add users to your feed" styling="tooltip-top button"> Top </button> <button tooltip="Add users to your feed" styling="tooltip-top-right button"> Top Right </button> <hr> <button tooltip="Add users to your feed" styling="tooltip-bottom-left button"> Bottom Left </button> <button tooltip="Add users to your feed" styling="tooltip-bottom button"> Bottom </button> <button tooltip="Add users to your feed" styling="tooltip-bottom-right button"> Bottom Right </button> <hr> <button tooltip="Add users to your feed" styling="tooltip-left button"> Left </button> <button tooltip="Add users to your feed" styling="tooltip-right button"> Right </button>
```

Colors

Tooltips can be in different colors

Example




```
<button tooltip="Add users to your feed" styling="tooltip-inverted button"> Black </button> <button tooltip="Add users to your feed" styling="tooltip-error tooltip-top-right button"> Error </button> <button tooltip="Add users to your feed" styling="tooltip-bottom-left tooltip-info button"> Info </button> <button tooltip="Add users to your feed" styling="tooltip-bottom tooltip-success button"> Success </button> <button tooltip="Add users to your feed" styling="tooltip-bottom-right tooltip-warning button"> Warning </button>
```

Typography

Default

Standard Button

Example

H1 - 2.18em

H2 - 1.64em

H3 - 1.45em

H4 - 1.33em

H5 - 1em

H6 - 0.83em

```
<h2>H1 – 2.18em</h2> <h2>H2 – 1.64em</h2> <h3>H3 – 1.45em</h3> <h4>H4 – 1.33em</h4>
<h5>H5 – 1em</h5> <h6>H6 – 0.83em</h6>
```

Small - Secondary

Example

App Studio Framework

App Studio Framework

App Studio Framework

App Studio Framework

App Studio Framework

App Studio Framework

```
<h2>App Studio <small>Framework</small></h2> <h2>App Studio
<small>Framework</small></h2> <h3>App Studio <small>Framework</small></h3> <h4>App
Studio <small>Framework</small></h4> <h5>App Studio <small>Framework</small></h5>
<h6>App Studio <small>Framework</small></h6>
```

Paragraph

Example

Lorem Ipsum is simply dummy text of the printing and typesetting industry. Lorem Ipsum has been the industry's standard dummy text ever since the 1500s
when an unknown printer took a galley of type and scrambled it to make a type specimen book. It has survived not only five centuries, but also the leap into electronic typesetting

```
<p>Lorem Ipsum is simply dummy text of the printing and typesetting industry. Lorem Ipsum has been the industry's standard dummy text ever since the 1500s</p> <p>when an unknown printer took a galley of type and scrambled it to make a type specimen book. It has survived not only five centuries, but also the leap into electronic typesetting</p>
```

Highlight

Can be highlighted

Example

Lorem Ipsum is simply dummy text of the printing and typesetting industry.

```
<p>Lorem Ipsum is simply <mark>dummy text</mark> of the printing and typesetting industry.</p>
```

FAQ

The articles in this section answer frequently asked questions about various aspects and functionality of App Studio.

More information

See the following how-to articles:

- *Adding Twitter and YouTube Results*
- *Changing the Color Scheme*
- *Changing the Logo*
- *Dynamically Adding Icons To Fields*
- *Removing the Security Module*
- *Setting up a Mock Platform Response*

App Studio Versions

How are these products versioned?

- **Appkit** is versioned according to updates to the code base.
- **App Studio Enterprise** versions are loosely tied to the version of the Appkit framework it includes. An update from **4.8.0** to **4.9.0** may not include any changes to App Studio Enterprise.

What about compatibility?

- Any version of **Appkit** can be used with Fusion 4.x. However, only Appkit 4.6.0+ can be used with Fusion 5.y.
- **App Studio Enterprise** can be used with any version of Fusion or Appkit.

Can these products be updated independently?

- **Appkit** can be *updated* independently, assuming there are no version *compatibility* issues.
- **App Studio Enterprise** is released when a new version of Appkit is released. As a result, App Studio Enterprise itself does not need to be updated. To move to the latest code base, simply update the Appkit version within the project.

Devices and Browser Support

What browsers can be supported? For example IE 11 or 9?

Any modern browser will be perfectly fine. For Internet Explorer, we recommend at least version 11. Appkit uses progressive enhancements so for older browsers the application will still work but in a feature limited fashion.

How does AppKit support responsive design?

All standard Appkit templates are responsively designed and will work correctly on tablets and mobiles with full feature parity.

What is the process we must follow to accomplish responsive design?

As long as the existing templates are not compromised, responsive design will work out of the box. The user interface being adapted is a customization of a fully responsive template.

Empty Facets in Solr

How do I avoid displaying empty facet values in my Appkit visualization?

By default Solr will return all the facet values for a given facet field, regardless of whether they match any documents in the result sets.

This behavior is controlled by the Solr facet.mincount configuration parameter *(https://cwiki.apache.org/confluence/display/solr/Faceting#Faceting-Thefacet.mincountParameter)* which defaults to `0`.

Setting the facet.mincount parameter to `1` will ensure that only values which match at least one document will be returned in the facet list.

This can be done by adding a custom parameter at query time or by modifying the search handler *(https://cwiki.apache.org/confluence/display/solr/RequestHandlers+and+SearchComponents+in+SolrConfig#RequestHandlersandSearchComponentsinSolrConfig-SearchHandlers)* in the `solrconfig.xml` file for the relevant collection.

Performance

What kind of performance testing has been done on Appkit and can results be shared?

Appkit is routinely profiled and performance tested. This includes load and capacity testing for concurrent users as well as time taken for requests to be processed. Through years of development and deployments each component will function to a very high performance specification as evidenced by several high load public facing sites including Vodafone, Financial Times, and Thomson-Reuters. Another Appkit customer TES Global (https://www.tes.co.uk *(https://www.tes.co.uk)*) experiences peak loads of 100 requests per second.

We track the performance of Appkit in numerous scenarios across releases to ensure there is no degradation in performance. In nearly every installation the performance bottleneck and largest proportion of time taken during the request-response cycle is due to the search platform. Tuning of the search platform and infrastructure, for example horizontal scaling is recommended in large scale deployments with concurrent users.

Is there a process to optimize performance?

The performance of a given application tends to be more bound by the search infrastructure. Appkit is normally only approximately 5-10% of the overall request lifecycle with the majority of the time being spent on IO within the search engine.

Appkit is a stateless system, so we advocate building out the application as per user requirements and then load testing that to expected loads x 3. Performance will vary more by features selected and infrastructure/network performance, but due to Appkit's stateless nature you can add as many instances as required.

Responsive Design

How does AppKit support responsive design?
All standard Appkit templates are responsively designed and will work correctly on tablets and mobiles with full feature parity.

What is the process we must follow to accomplish responsive design?
As long as the existing templates are not compromised, responsive design will work out of the box. The user interface being adapted is a customization of a fully responsive template.

Space for Log Files

How much space should you allow for Appkit logs?

Less than 50 MB for Appkit logs should be more than sufficient in production, after you reduce the log levels to INFO in `logback.xml`.

However this is entirely configurable. You can set up rolling daily logs, only holding back the last X. See the Logback documentation for more information *(http://logback.qos.ch/manual/appenders.html)*.

Deployment

This article describes general app deployment with App Studio.

Fusion Webapps service

Fusion Web app deployment gives you an embedded instance of App Studio in each of your Fusion instances.

Standalone deployments

- *Java WAR file for deployment in a servlet container*
  Package your search app in a Java WAR file, which can be deployed in a servlet container such as Tomcat or Jetty. This is the most common way to deploy a search app.
- *Java JAR file with embedded Tomcat web server*
  Package your search app in a Java JAR file that also contains an embedded Tomcat web server. The app is ready to deploy in a production environment. It is not necessary to build a WAR file and deploy that in a servlet container or application server.
- *Executable Java JAR file for execution in a Java runtime environment*
  Package your search app in a Java JAR file. You can run the app with the Java launcher.

Building from Source

Appkit can either be deployed as a simple web application archive (.war) file — *see Installing Appkit* — or as a Java Web Application built using Maven *(http://maven.apache.org)*, as described in this article.

Install Java 7+ JDK, Git and Maven for your platform

1. Download *(https://www.oracle.com/java/technologies/javase-jdk8-downloads.html)* and install the Oracle (SUN) Java SE JDK 8.
2. Install npm 5.0+ *(https://www.npmjs.com/get-npm)*.
3. Configure Maven by saving the following `settings.xml` file in your Maven configuration folder within your App Studio Enterprise project (`/bin/settings.xml`). See the Maven Settings Reference *(http://maven.apache.org/settings.html)* for more information.

```
<?xml version="1.0"?> <!-- Credentials for Twigkit repositories --> <settings
xmlns="http://maven.apache.org/SETTINGS/1.1.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/SETTINGS/1.1.0
http://maven.apache.org/xsd/profile-1.1.0.xsd"> <servers> <server>
<id>twigkit.com</id> <username>evaluator</username>
<password>AP6YhQQWq5cqu85Eu5JwMZnJk48</password> </server> <server>
<id>twigkit.com-releases</id> <username>evaluator</username>
<password>AP6YhQQWq5cqu85Eu5JwMZnJk48</password> </server> <server>
<id>twigkit.com-snapshots</id> <username>evaluator</username>
<password>AP6YhQQWq5cqu85Eu5JwMZnJk48</password> </server> </servers> <profiles>
<profile> <id>twigkit-default</id> <!-- Artifact repositories --> <repositories>
<repository> <id>twigkit.com</id> <name>Twigkit</name>
<url>https://twigkit.jfrog.io/twigkit/repo</url> <releases> </releases>
<snapshots> </snapshots> </repository> </repositories> <!-- Plugin repositories
--> <pluginRepositories> <pluginRepository> <id>twigkit.com</id>
<name>Twigkit</name> <url>https://twigkit.jfrog.io/twigkit/repo/</url>
<releases> <enabled>true</enabled> <updatePolicy>never</updatePolicy>
<checksumPolicy>warn</checksumPolicy> </releases> <snapshots>
<enabled>true</enabled> <updatePolicy>always</updatePolicy>
<checksumPolicy>warn</checksumPolicy> </snapshots> </pluginRepository>
</pluginRepositories> </profile> </profiles> <activeProfiles>
<activeProfile>twigkit-default</activeProfile> </activeProfiles> </settings>
```

Download App Studio Enterprise

Go to the *App Studio Enterprise download page* and download the latest copy.

Run the web application

After you have downloaded the Git project, you can run the application using the included start scripts. Enter this command from the project folder:

```
./app-studio start
```

Or, for Windows:

```
app-studio.bat start
```

After the application has started, you can access it from your browser at `http://localhost:8080/`.

Package the web application

To package the web application for deployment (rather than run it locally), you can run this command from the project folder:

```
./app-studio package
```

Or, for Windows:

```
app-studio.bat package
```

This will generate a web application archive file (.war) in the target/ folder. You can then deploy the generated package as described in *Deploying an Appkit .war file*.

Appkit also supports creating self-starting applications which bundle the application server. When you package the application a self-starting archive is also created:

In the `/target` folder you will find both the WAR file and the self-starting one, for example, `app-studio-enterprise.jar` (note the .jar suffix). To run the application use:

```
java -jar app-studio-enterprise.jar
```

Use build profiles to package the web application for different runtime environments

Often the only differences between applications deployed to different environments are in the resources, that is, configuration settings, security configuration, or in the license key that is used. To facilitate these differences Appkit also provides the capability to configure which runtime environment you want to deploy to when packaging an application for distribution.

To configure an application for different environments, the resources to be deployed to a particular environment are stored in `src/main/profiles/<environment>` in the source code tree. The assumption being that the main source tree will continue to contain all the resources intended for the `production` build of the application (that is, `src/main/resources`). For example, to configure an application for use in three different environments, in this case `dev`, `staging`, and eventually `production`, the source tree could be set up as follows:

```
/src /main /profiles /dev /conf spring-security.xml twigkit.lic /staging
twigkit.lic /java /resources << here is where the resources will be stored for the
production build
```

To package an application with a particular build profile, the profile name is given with the `-P` flag. For example, to deploy to the `dev` environment you would use this command:

```
./app-studio package -P dev
```

Prior to packaging, the contents of the build profile directory are merged with the resources under the main resources directory, overwriting any existing files, and including any additional files. This does not change any of the resources stored under the main resources directory, intended for the `production` build. The profile resources are only merged with main resources in the pre-packaging phase of the build after they are in the `target/classes` directory.

After the application has been packaged, the war and jar files associated with that particular application profile will be stored under the application's `dist/<environment>` directory where `<environment>` is the name given to the build profile. For example:

```
/dist /dev <application-name>.jar <application-name>.war /staging <application-name>.jar <application-name>.war <application-name>-1.0-SNAPSHOT.jar << both this and the war file below are for the production build <application-name>.war
```

Here, you can see that the application has been packaged for three different environments. One for the `dev` environment, another for the `staging` environment, and finally one for the `production` environment. The `<application-name>` would be replaced by the name of the application. The application version number is appended to the production jar file.

To test a profile locally, you can use its self-starting archive JAR file. To run the application, navigate to the directory the JAR file is in and use this command:

```
java -jar <application-name>.jar
```

Again replacing `<application-name>` with the actual name of the application.

Develop a Maven build profile by overlaying configurations

Together with configuring different build profiles for different runtime environments, Appkit also offers the ability to *overlay configurations* via the Jetty Maven plugin. This can be very useful when you want to test whether specific values can be used in the build profile prior to deployment.

By default, when a build profile is run via the Jetty Maven plugin, a default overlay will also be used if it is placed in `src/dev/resources/conf`:

```
mvn clean jetty:run -Pdevelopment -Dtwigkit.profile=dev
```

However, if the path to the overlay is different it can still be picked up by adding the system parameter `-Dtwigkit.conf.overlay`:

```
mvn clean jetty:run -Pdevelopment -Dtwigkit.profile=dev -Dtwigkit.conf.overlay="file://src/dev/my_resources/conf"
```

In terms of merge order, the build profile is merged first followed by the overlay. So you will see in the final loaded configuration that the overlay takes precedence.

> The overlay is purely meant for development. It will not be used when an app is deployed (via `mvn clean package`). During this phase, any resource under the build profile will overwrite the equivalent resource in the main app.

Search Platforms

Appkit integrates with a number of search engines, data warehouses and web service endpoints, even surfacing information from multiple sources simultaneously. The platform abstraction covers both search and storage sides, meaning that Appkit will implement for each platform the necessary protocols to save or index content for any platform. A number of platform adapters are available out-of-the-box with Appkit, and *adding new ones* is a simple exercise.

This is a summary of all types of data source currently supported by Appkit. Each data source has an associated set of connectors, that are responsible for brokering communications between the end-user and the underlying search endpoint. In each case, the corresponding search platform is typically configured through platform configuration files, all of which share a number of *common attributes*.

Search engines

- *Fusion*
- *Solr*
- *SolrCloud*
- *Elasticsearch*

Other data sources

- *Wolfram|Alpha*
- *Wikipedia*
- *Pipl*
- *YouTube*
- *Twitter*
- *Google Custom Search*

Related how-to articles

- *Fusion Impersonation via a Service Account*
- *Set Up Active Directory Federation Services (ADFS) Authentication*
- *Set Up Facebook Authentication*
- *Set Up Google Authentication*
- *Set Up Office 365 Authentication*

Solr

The Solr search adapter brokers all communication with the Solr search engine *(http://lucene.apache.org/solr/)*. To work with SolrCloud deployments, read about our *SolrCloud* search adapter.

Setup

To add Solr search connectors to a Maven project, you must add this dependency to your project's `pom.xml` file:

```
<dependency> <groupId>twigkit</groupId>
<artifactId>twigkit.solr.runtime</artifactId> <version>${project.parent.version}
</version> </dependency>
```

Usage

Assuming you have added the correct dependencies to your project (see above), the Solr platform configuration can be accessed by pointing to it in the search:platform tag. In this case the platform is placed in a variable named `platform` using the `var` attribute, and points to a platform configuration in ../conf/platforms/solr/solr.conf:

```
<search:platform var="platform" conf="platforms.solr"></search:platform>
```

platforms/solr/solr.conf

In this example platform configuration, we set some general settings for the platform:

```
name: twigkit.search.solr.Solr resultIDField: id defaultQuery: *:* url:
http://100.200.300.400:8983/solr/myCollection defaultFacets: age,weight
```

Required attributes

`url (java.lang.String)`
The URL of the Solr Server. Must be specified using the full URL (for example, `http://localhost:8983/solr/collection` - `localhost:8983` will not work). If a value for `url` is found in the configuration, this will take precedence over other parameters found for `protocol`, `host`, `port`, and `collection`.

`protocol (java.lang.String)` The protocol used to send requests to the Solr Server, for example, `https` in `https://localhost:8983/solr/collection`.
Default: `http`

`host (java.lang.String)`
The domain name of the Solr server, for example, `localhost` in `http://localhost:8983/solr/collection`.

Use of `host` as the URI of the Solr Server is deprecated.

`port (java.lang.String)`
The port used to access the Solr Server, for example, `8983` in `https://localhost:8983/solr/collection`.
Default: 8983

`collection (java.lang.String)`
The Solr collection being accessed, for example, `collection` in `https://localhost:8983/solr/collection`.

As mentioned above for the `host` attribute, if a non-empty value for `url` is found in the configuration, this will take precedence over other attributes found for `protocol`, `host`, `port`, and `collection`. If no `url` attribute is found, and a value for `host` is found

that begins with `http`, `https`, `file` or `/` then this will be taken as the full URL. Finally, if the latter condition is also not met, the URL will be built from attributes `protocol`, `host`, `port`, and `collection` if present.

Optional attributes

`requestHandler (java.lang.String)`
Name of the Request Handler to use. Use this to specify an alternate Request Handler to use for example, the '/dismax' request handler. Note you can override this by specifying a 'custom' attribute on the query.

`requestMethod (java.lang.String)`
Which HTTP request method to use (GET or POST).

`strategy (java.lang.String)`
Whether to cluster multiple hosts (value should be either combine, or round-robin). To enable sharding, set strategy to combine.

`groupField (java.lang.String)`
Setting this enables Solr Result Grouping or Field Collapsing on the given field. This will create a Result for each Group returned. Matching documents for the group will be available as related results.

`timeOut (java.lang.Integer)`
Time out for platform query requests (in milliseconds).
Default: 5000

`ignoreAppliedFilters (java.lang.Boolean)`
If this is set to false, facet filters that have already been applied to the Query are added to the list of filters for the facet.
Default: true

`backwardsCompatible (java.lang.Boolean)`
For Solr 1.x (for example, 1.4) set backwardsCompatible to true.
Default: false

`highlight (java.lang.Boolean)`
Whether to enable hit highlighting (hl parameter) in Solr.
Default: true

Attributes shared across platforms

`name (java.lang.String)` The display name of this Platform instance. Used for identifying different platforms in the user interface.

`aliases (java.lang.String)`
Manage mappings from field names to aliases (use a comma-separated list). All references to the field (via this platform) in results, facets, filters, and query strings are mapped. For example, to refer to a field in the index named `firstnamelastname` as `name` and `countryofresidence` as `country`, use these mappings:
`aliases="firstnamelastname=name,countryofresidence=country"`

`defaultQuery (java.lang.String)`
Default query to use when none is specified. For Solr use `:` to bring back all items (for example, for 'zero term search').

`defaultFacets (java.lang.String)`
Default facets to request when none are specified. This is a comma separated list.

`spellCheck (java.lang.Boolean)`
Spellcheck the Query term if supported by the Platform.
Default: `true`

`expandQuery (java.lang.Boolean)`
Apply advanced linguistics such as stemming or lemmatization if supported by the Platform.
Default: `true`

`autoCorrect (java.lang.Boolean)`
Whether to auto-correct and resubmit futile queries (queries with zero results).
Default: `true`

`fileTypeField (java.lang.String)`
Set which field contains information about file type (mime type).

`resultIDField (java.lang.String)`
Set which field represents the unique identifier for a given result. Should correspond to the <uniqueKey> element in the Solr schema.xml.

`pageLimit (java.lang.Long)`
A limitation to which page the platform will offset.

Hit highlighting

Hit highlighting is enabled by default, and assumes that the uniques uniqueKey field in the Solr `schema.xml` file is named id. If your uniqueKey is set to a different field highlighting will not work unless you set the resultIDfield key in the Appkit configuration to the right field.

```
resultIDField = yourUniqueKey
```

Appkit expects that Solr's default highlighting tags of `<em></em>` are used. Ensure that this has not been set to something different using `hl.simple.pre` and `hl.simple.post` settings in the Solr or request handler configurations.

Configuring the Hierarchical Facet Service

To correctly display a taxonomy as a hierarchical facet you must configure the requisite Appkit service to retrieve the data structure that is configured in your Solr index.

First you must create a service configuration:

```
/conf /services /hierarchical /hierarchical.conf
```

The file 'hierarchical.conf' should be configured similar to this example:

```
name: twigkit.search.solr.service.facet.GreedySolrHierarchicalFacetService
platform: platforms.solr parents-suffix: _parent
```

`platform`
This specifies the configured platform where the taxonomy has been indexed.

`parents-suffix`
The suffix that, if appended to the facet's fieldname, designates the field containing parent node data. The suffix will depend on how the taxonomy data has been indexed.

This service configuration is what is used when adding the hierarchical facet to your HTML page as in this example:

```
<search:facet facet-name="location" show="6" show-more="100" max-characters="35"
collapsible="true"> <facet:hierarchical title="location" facet-name="location"
platform="myHierarchicalPlatform" query="myHierarchicalQuery"></facet:hierarchical>
</search:facet>
```

Indexing Hierarchical Taxonomies

For efficient hierarchical faceting in Solr you should index the nodes in your hierarchy as a multi-valued field using a depth (or level) prefix. This follows Solr best practice for hierarchical faceting, as described here:

http://wiki.apache.org/solr/HierarchicalFaceting *(http://wiki.apache.org/solr/HierarchicalFaceting)*

As an example, suppose you have three documents with these values for the `location` field:

```
Doc#1: Europe > Germany > Berlin Doc#2: Europe > France > Paris Doc#3: North America > United States > California > San Francisco
```

These three documents fall into this taxonomy:

```
Europe (Doc#1, Doc#2) | | + France (Doc#1) | | | +– Paris (Doc#1) | +– Germany (Doc#2) | +– Berlin (Doc#2) North America (Doc#3) | +– United States (Doc#3) | + California (Doc#3) | +– San Francisco (Doc#3)
```

To represent this taxonomy structure in your index, you would add these values for the (multivalued) `location` field:

```
Doc#1: location: 0/Europe 1/Europe/Germany 2/Europe/Germany/Berlin Doc#2: location: 0/Europe 1/Europe/France 2/Europe/France/Paris Doc#3: location: 0/North America 1/North America/United States 2/North America/United States/California 3/North America/United States/California/San Francisco
```

The way to read this is that Doc#1 matches the `Europe` level-0 (or root) category *and* the `Europe > Germany` level-1 category, and so forth. Through this indexing scheme, you can get facet counts for all documents that belong to the `Europe > Germany` category (at level 1) by querying for "1/Europe/Germany" as a Solr facet prefix, but equally all documents that as a result belong to `Europe` at a broader level. Finally this allows us to also get facet counts for all documents with any `location` value by requesting "0/" as a facet prefix, for example.

Optimising the retrieval of child nodes

When you display a hierarchical facet in Appkit, Appkit initially only shows the top-level nodes in the taxonomy and asynchronously fetch nodes further down the tree by using Solr facet prefix queries as each node gets expanded. This greatly reduces the size of the taxonomy that would have to be fetched initially, if you had to request the whole tree and render it on the page all at once. This means that, irrespective of how deep and wide your taxonomy is, it can be represented in the user interface in a performant manner.

However, when using only one categorisation field, Appkit does not know whether to show a link to expand each node unless it looks ahead and does another Solr query to check whether a given node has any children. To address this, you can further augment the information Appkit indexes for each document that is tagged with hierarchical categories, so that Appkit can quickly look up both a single tree level (for example, `facet.prefix = 0/`) and also determine which of the nodes at a given level have children (using a single facet query).

More specifically, for each hierarchical facet `my_facet`, you create an additional meta-facet named `my_facet_parents` that contains information about the taxonomy categories that have children (that is, those nodes that are not leaf nodes in the hierarchy). Using the example from above, you would index `location` in exactly the same way as before. In addition, you would index these terms for the new `location_parents` meta-facet:

```
Doc#1: location: 0/Europe 1/Europe/Germany 2/Europe/Germany/Berlin
location_parents: 0/Europe 1/Europe/Germany Doc#2: location: 0/Europe
1/Europe/France 2/Europe/France/Paris location_parents: 0/Europe 1/Europe/France
Doc#3: location: 0/North America 1/North America/United States 2/North
America/United States/California 3/North America/United States/California/San
Francisco location_parents: 0/North America 1/North America/United States 2/North
America/United States/California
```

> The new `location_parents` field has all the same index terms as `location` except *the last level is omitted*. The intended meaning is that Doc#1 is indexed with the term `0/Europe` in `location_parents` because it contains `location` values that are the children of `Europe`, and so forth.

With this scheme in place, Appkit will automatically generate and run a single facet query to retrieve all top-level nodes and identify those nodes that have children.

Query Suggestions

Solr provides the means to suggest documents or terms to the end-user as he is typing search terms in the search box. It can suggest terms based on a dictionary file, terms indexed in a specific field or suggest entire documents based on the occurrence of the search terms in a specific field. For more details on how to implement these features, refer to the suggester *(https://cwiki.apache.org/confluence/display/solr/Suggester)* as well as the terms component *(https://cwiki.apache.org/confluence/display/solr/The+Terms+Component)*.

Suggester

As an example here, these are things that you must add to your `solrconfig.xml` file to get suggestions based on matching against a `title` field, as well as looking up terms in your main `content` field.

```
<searchComponent name="suggest" class="solr.SuggestComponent"> <!--Suggestion based on matches in the title. This will return the entire title field as a suggestion--> <lst name="suggester"> <str name="name">titleMatchSuggester</str> <str name="lookupImpl">FuzzyLookupFactory</str> <str name="dictionaryImpl">DocumentDictionaryFactory</str> <str name="field">title</str> <str name="suggestAnalyzerFieldType">text_general</str> <str name="buildOnStartup">true</str> </lst> <!--Suggestion based on keyword matches in the content field. This will only return the individual words matching the input string--> <lst name="suggester"> <str name="name">freeTextContentSuggester</str> <str name="lookupImpl">FreeTextLookupFactory</str> <str name="dictionaryImpl">DocumentDictionaryFactory</str> <str name="field">content</str> <str name="suggestFreeTextAnalyzerFieldType">text_general</str> <str name="suggestAnalyzerFieldType">text_general</str> <str name="buildOnStartup">true</str> </lst> </searchComponent> <requestHandler class="org.apache.solr.handler.component.SearchHandler" name="/suggest"> <lst name="defaults"> <str name="suggest">true</str> <str name="suggest.count">5</str> <str name="suggest.dictionary">titleMatchSuggester</str> <str name="suggest.dictionary">freeTextContentSuggester</str> </lst> <arr name="components"> <str>suggest</str> </arr> </requestHandler>
```

After this is set up and you have restarted your Solr engine, the query `http://localhost:8983/solr/sample/suggest?q=al` (replace 'al' with the string the end-user has typed) will bring back suggestions for both suggesters in this format:

```
<response> <lst name="responseHeader"> <int name="status">0</int> <int name="QTime">3</int> </lst> <lst name="suggest"> <lst name="titleMatchSuggester"> <lst name="al"> <int name="numFound">1</int> <arr name="suggestions"> <lst> <str name="term">allpicsppt</str> <long name="weight">0</long> <str name="payload"/> </lst> </arr> </lst> </lst> <lst name="freeTextContentSuggester"> <lst name="al"> <int name="numFound">4</int> <arr name="suggestions"> <lst> <str name="term">all</str> <long name="weight">14243018808291072</long> <str name="payload"/> </lst> <lst> <str name="term">alternative</str> <long name="weight">4069433945226020</long> <str name="payload"/> </lst> <lst> <str name="term">aligned</str> <long name="weight">2034716972613010</long> <str name="payload"/> </lst> <lst> <str name="term">alzheimer’s</str> <long name="weight">2034716972613010</long> <str name="payload"/> </lst> </arr> </lst> </lst> </lst> </response>
```

This response basically suggests one document with a title that contains the keyword typed by the end user, as well as suggesting search terms that exist in the index.

Accessing the Solr Suggest component in Appkit

Appkit provides two different services to access Solr suggestions depending on whether or not backwards compatibility (set via the `backwardsCompatible` attribute) is enabled.

Backwards compatibility is disabled

When backwards compatibility is disabled, use the `SolrQuerySuggestionService` to access suggestions. In `resources/conf/services/suggestions`, create a configuration file (for example, `querySuggestions.conf`) to contain the configuration parameters for the suggestion service. Here is an example of how this might look:

```
name: twigkit.search.solr.suggestions.SolrQuerySuggestionService source:
platforms.solr parenthesis: true title: Staff custom: count[5] doctype: reference
```

Both the `name` and `source` attribute are required. The `name` indicates which suggestion service is used. The `source` gives the id for accessing the platform configuration attributes.

The remaining attributes shown here are optional. If `parenthesis` is set to true, the query value, equated with the URL parameter `suggest.q`, will be surrounded by parenthesis. The `title` attribute gives a title to the list of suggestions. Custom parameters can also be set on the query. For example, `count[5]` translates to `count=5` on the query URL.

Backwards compatibility is enabled

When backwards compatibility is enabled, use the `SolrSuggestionsParserService` to access suggestions. In `resources/conf/services/suggestions` create a configuration file (for example, `querySuggestions.conf`) to contain the configuration parameters for the suggestion service. Here is an example of how this might look:

```
name: twigkit.search.solr.suggestions.SolrSuggestionsParserService source:
platforms.fusion suggester: titleMatchSuggester parenthesis: true title: Staff
custom: count[5] doctype: reference
```

As above, both the `name` and `source` attributes are required. The `source` attribute here refers to a Fusion platform configuration, which must have the `backwardsCompatible` attribute set to true. The other required attribute for this service is `suggester`. This is the name of the Solr suggester as defined in the solrconfig.xml. For example, using the example above, `suggester` could be set to titleMatchSuggester or freeTextContentSuggester.

The remaining attributes shown here are optional and are the same as described above for when backwards compatibility is disabled.

Terms component

The Solr terms component *(https://cwiki.apache.org/confluence/display/solr/The+Terms+Component)* "provides access to the indexed terms in a field and the number of documents that match each term". This can be used, for example, to find all terms in the Solr index that match an expansion of the given query term. For example, "car" might return CARRY, CARRIED, CARE, CARL etc.

Accessing the Solr Terms component in Appkit

To access the Terms component in Appkit, use the `SolrQueryTermsService`. In `resources/conf/services/suggestions` create a configuration file (for example, `queryTerms.conf`) to contain the configuration parameters for the terms suggestion service. Here is an example of how this might look:

```
name: twigkit.search.solr.suggestions.SolrQueryTermsService source:
platforms.solr.suggestions.terms term-fields: Symbol title: Suggested phrase
doctype: reference
```

Both the `name` and `source` attributes are required as well as the `term-fields` attribute. The `name` indicates which suggestion service is used. The `source` gives the configuration name for accessing the Solr platform's configuration attributes (in the example shown above the platform configuration would be stored in `resources/conf/platforms/solr/suggestions/terms.conf`. Within this platform configuration the `requestHandler` attribute mat be defined. However, by default, this is set to `/terms`.

The `term-fields` attribute is equivalent to the `terms.fl` parameter required by the Solr Terms Component. It is a comma-separated list of fields from which to retrieve terms. In the example above, only one field, `Symbol`, is used.

The `title` attribute gives a title to the group of suggestions that are returned by Solr. This title can be used to group suggestions in the UI.

Referring to the suggestion and/or terms service in the search:box

Finally, to ensure your application hooks into the suggestion and/or terms service when it is run, place this line inside your `<search:box>`:

```
<query:suggestions completion-service="services.suggestions.querySuggestions"
action="/search"></query:suggestions> <query:suggestions completion-
service="services.suggestions.queryTerms" action="/search"></query:suggestions>
```

With this in place, when you type a query into the search box, a drop-down list of query suggestions and/or terms will be shown that you can click on to fire a query.

SolrCloud

The SolrCloud search module brokers all communication with a SolrCloud cluster *(https://cwiki.apache.org/confluence/display/solr/SolrCloud)*. See also documentation on the Solr[Solr platform] for further background.

Setup

To add SolrCloud search connectors to a Maven project, you must add this dependency to your project's `pom.xml` file:

```
<dependency> <groupId>twigkit</groupId> <artifactId>twigkit.solr.cloud</artifactId>
<version>${project.parent.version}</version> </dependency>
```

Usage

Assuming you have added the correct dependencies to your project (see above), the Solr platform configuration can be accessed by pointing to it in the search:platform tag. In this case the platform is placed in a variable named `platform` using the `var` attribute, and points to a platform configuration in ../conf/platforms/solr/solr.conf:

```
<search:platform var="platform" conf="platforms.solr"></search:platform>
```

platforms/solr/solr.conf

In this example platform configuration, we set some general settings for the platform:

```
name: twigkit.search.solr.cloud.SolrCloud resultIDField: id defaultQuery: *:*
zookeeper-host: cld-dapp-slr61:2181,cld-dapp-slr62:2181,cld-dapp-slr63:2181
defaultFacets: age,weight default-collection: mycollection
```

Required attributes

`zookeeper-host (java.lang.String)`
The client endpoint of the zookeeper quorum containing the cloud state. The full specification for this string is one or more comma separated `HOST:PORT` values, followed by an optional `chroot` value that starts with a forward slash. Using a `chroot` allows multiple applications to coexist in one ensemble. For full details, see the ZooKeeper documentation *(https://zookeeper.apache.org/doc/r3.2.2/zookeeperAdmin.html)*.

Here are some examples:

- `host1:2181`
- `host1:2181,host2:2181,host3:2181/mysolrchroot`
- `zoo1.example.com:2181,zoo2.example.com:2181,zoo3.example.com:2181`

Optional attributes

`groupField (java.lang.String)`
Setting this enables Solr Result Grouping or Field Collapsing on the given field.

`backwardsCompatible (java.lang.Boolean)`
For Solr 1.x (for example, 1.4) set backwardsCompatible to true.
Default: `false`

`ignoreAppliedFilters (java.lang.Boolean)`
If this is set to false, facet filters that have already been applied to the Query are added to the list of filters for the facet.

Default: `true`

`requestMethod (java.lang.String)`
Which HTTP request method to use (GET or POST).

`highlighting (java.lang.Boolean)`
Whether to enable hit highlighting (hl parameter) in Solr.
Default: `true`

`max-query-resend-attempts (java.lang.Integer)`
The maximum number of times a query is re-sent after a ZooKeeper session expires.
Default: 1

Attributes shared across platforms

`default-collection (java.lang.String)`
Specifies the default Solr collection for the request.

`name (java.lang.String)`
The display name of this Platform instance. Used for identifying different platforms in the user interface.

`aliases (java.lang.String)`
Manage mappings from field names to aliases (use a comma-separated list). All references to the field (via this platform) in results, facets, filters, and query strings are mapped. For example, to refer to a field in the index named `firstnamelastname` as `name` and `countryofresidence` as `country`, use these mappings:
`aliases="firstnamelastname=name,countryofresidence=country"`

`defaultQuery (java.lang.String)`
Default query to use when none is specified.

`defaultFacets (java.lang.String)`
Default facets to request when none are specified. This is a comma separated list.

`spellCheck (java.lang.Boolean)`
Spellcheck the Query term if supported by the Platform.
Default: `true`

`expandQuery (java.lang.Boolean)`
Apply advanced linguistics such as stemming or lemmatization if supported by the Platform.
Default: `true`

`autoCorrect (java.lang.Boolean)`
Whether to auto-correct and resubmit futile queries (queries with zero results).
Default: `true`

`fileTypeField (java.lang.String)`
Set which field contains information about file type (mime type).

`resultIDField (java.lang.String)`
Set which field represents the unique identifier for a given result. Should correspond to the <uniqueKey> element in the Solr `schema.xml`.

`pageLimit (java.lang.Long)`
A limitation to which page the platform will offset.

Use of the fields parameter in the Appkit query tag with the SolrCloud platform

By default, on all platforms, if the optional `fields` parameter is not used in the `<search:query>` tag, all fields will be returned in the response. This is not the case however with the SolrCloud platform. On this platform, if the `fields` parameter is omitted no fields will be returned in the response. This is to work around an issue identified with the SolrCloud platform that does not allow fields given by Appkit to be overridden by those defined in an invariants section of the SolrCloud request handler. To return all fields in the case, the fields parameter would must contain a wildcard character, `fields="*"`.

Elasticsearch

Setup

To add Elasticsearch connectors to a Maven project, you must add one of the following dependencies to your project's `pom.xml` file, depending on which version of Elastic you are using:

- For pre-V5 Elasticsearch, add this dependency:

```
<dependency>
    <groupId>twigkit</groupId>
    <artifactId>twigkit.elasticsearch.v2</artifactId>
    <version>${project.parent.version}</version>
</dependency>
```

- For Elasticsearch v5, add this dependency:

```
<dependency>
    <groupId>twigkit</groupId>
    <artifactId>twigkit.elasticsearch.v5</artifactId>
    <version>${project.parent.version}</version>
</dependency>
```

- For Elasticsearch v6, add this dependency:

```
<dependency>
    <groupId>twigkit</groupId>
    <artifactId>twigkit.elasticsearch.v6</artifactId>
    <version>${project.parent.version}</version>
</dependency>
```

For projects that are currently using the non-version-numbered dependency below, we recommend that you update the dependency to one of the version-numbered ones above.

```
<dependency> <groupId>twigkit</groupId>
<artifactId>twigkit.elasticsearch</artifactId> <version>${project.parent.version}
</version> </dependency>
```

Usage

This example shows how to configure an Elasticsearch platform on a JSP page.

```
<search:platform var="platform" conf="platforms.elastic"></search:platform>
```

This is an example platform configuration that would be at platforms/elastic/elastic.conf, with a number of attributes exposed:

```
name: twigkit.search.elasticsearch.ElasticSearch timeOut: 30000 resultIDField: id
highlight: true defaultQuery: *:* # Elasticsearch host in the format
my.organisation.com host: localhost # Elasticsearch TCP port (defaults to 9300)
#port: # Elastic cluster cluster: elasticsearch # Elastic index index: test
#Security Settings for Shield #user-name: #password:
```

Required attributes

`host (java.lang.String)`
The URI of the ElasticSearch Server. Must be specified using the full URI (for example, `http://localhost:1234`; `localhost:1234` will not work).

`preserveBoosting (java.lang.Boolean)`
If true, ensures the query is 'rewritten' by Elasticsearch so that field boosting is preserved.
Default: `false`

Optional attributes

`clientTransportSniff (java.lang.Boolean)`
Whether to sniff for other nodes on the same cluster allowing for more fault-tolerant and efficient search. Might be required if the Elasticsearch instances are distributed.
Default: `false`

`cluster (java.lang.String)`
The name of the cluster to use (required if using `clientTransportSniff="true"` for sniffing nodes on the cluster allowing for more fault-tolerance and efficient searching)
Default: `''`

`type (java.lang.String)`
The name of the data type to use, which will be defined by a mapping of fields in the Elasticsearch instance
Default: `''`

`index (java.lang.String)`
The search index, for example, 'search' for an index named search.
Default: `''`

`highlight (java.lang.String)`
Fields to apply highlighting to. Specified in a comma separated list.

`facetSize (java.lang.String)`
Determines the total number of all facets to be returned - the Elasticsearch default is 10 which is too small for most applications.

`applyFiltersToFacets (java.lang.Boolean)`
If true: facets returned will be restricted by the filters applied in the query so will match only those with matching documents in the result set.
Default: true

`shard-size (java.lang.Integer)`
For use with terms aggregations, this parameter decides how many terms the co-ordinating node will request from each shard.

`min-doc-count (java.lang.Long)`
For use with terms aggregations, terms will only be returned if they match more than this number of hits.

`shard-min-doc-count (java.lang.Long)`
For use with terms aggregations, terms will only be considered if their local shard frequency is higher than this value.

`show-term-doc-count-error (java.lang.Boolean)`
For use with terms aggregations, setting this value to true will return an error value for each term, which represents the worst case error in the document count.

`fields-type (java.lang.String)`
Available with Elastic Search 5 adapter only. Setting this value to `source` will enable the use of source filtering as described here *(https://www.elastic.co/guide/en/elasticsearch/reference/current/search-request-source-filtering.html)*. Default value is `stored` and uses the legacy stored_fields filtering described here *(https://www.elastic.co/guide/en/elasticsearch/reference/current/search-request-stored-fields.html)*.

`query-term-fields (java.lang.String)`
Available with Elastic Search 5 adapter only. The query can also search against multiple fields. These fields can be provided via the "query-term-fields" parameter here *(https://www.elastic.co/guide/en/elasticsearch/reference/5.2/query-dsl-query-string-query.html#_multi_field)*. Default value is `[]` that searches against all fields.

Attributes shared across platforms

`name (java.lang.String)`
The display name of this Platform instance. Used for identifying different platforms in the user interface.

`aliases (java.lang.String)`
Manage mappings from field names to aliases (use a comma-separated list). All references to the field (via this platform) in results, facets, filters, and query strings are mapped. For example, to refer to a field in the index named `firstnamelastname` as `name` and `countryofresidence` as `country`, use these mappings:
`aliases="firstnamelastname=name,countryofresidence=country"`

`defaultQuery (java.lang.String)`
Default query to use when none is specified.

`defaultFacets (java.lang.String)`
Default facets to request when none are specified. This is a comma separated list.

`spellCheck (java.lang.Boolean)`
Spellcheck the Query term if supported by the Platform.
Default: `true`

`expandQuery (java.lang.Boolean)`
Apply advanced linguistics such as stemming or lemmatization if supported by the Platform.
Default: `true`

`autoCorrect (java.lang.Boolean)`
Whether to auto-correct and resubmit futile queries (queries with zero results).
Default: `true`

`fileTypeField (java.lang.String)`
Set which field contains information about file type (mime type).

`resultIDField (java.lang.String)`
Set which field represents the unique identifier for a given result.

`pageLimit (java.lang.Long)`
A limitation to which page the platform will offset.

Security with Elastic Shield

You might use the Elastic Shield plugin to Elasticsearch for securing access to your Elasticsearch infrastructure. You can easily configure the Elasticsearch search platform to handle this.

To add an Elasticsearch search platform that is secured by Elastic Shield
1. Add this dependency to your `pom.xml` file:

```
<dependency> <groupId>twigkit</groupId>
<artifactId>twigkit.elasticsearch.shield</artifactId>
<version>${project.parent.version}</version> </dependency>
```

2. Configure the credentials for accessing Elasticsearch secured by the Shield plugin. Specify `user-name` and `password` for the config definition.

```
** `*user-name (java.lang.String)*` + The userName to supply ** `*password
(java.lang.String)*` + The password to supply
```

Security with Elastic X-Pack

	This functionality is available in post-v5 Elasticsearch.

You might use Elastic X-Pack for securing access to your Elasticsearch infrastructure. You can easily configure the Elasticsearch search platform to handle this.

To add an Elasticsearch search platform that is secured by Elastic X-Pack
Configure the credentials for accessing Elasticsearch secured by Elastic X-Pack. Specify `user-name` and `password` for the config definition.

- **`user-name (java.lang.String)`**
  The userName to supply
- **`password (java.lang.String)`**
  The password to supply

Google Custom Search

The Google Custom Search Engine (CSE) platform module brokers all communication with a Google Custom Search Engine *(https://cse.google.co.uk/cse/)*.

Setup

To add Google CSE connectors to a Maven project, you must add this dependency to your project's `pom.xml` file:

```
<dependency> <groupId>twigkit</groupId> <artifactId>twigkit.google.cse</artifactId>
<version>${project.parent.version}</version> </dependency>
```

Required attributes

`apiKey (java.lang.String)`
The unique ID of the CSE (much like an API key).

Optional attributes

`client (java.lang.String)`
The CSE 'client' query parameter - which front-end to use.

`dateFields (java.lang.String)`
Which fields in the index represent dates. This is a comma separated list.

`numericFields (java.lang.String)`
Which fields in the index represent numbers. This is a comma separated list.

`output (java.lang.String)`
Return output with (xml) or without (xml_no_dtd) Google's DTD.
Default: `xml_no_dtd`

Attributes shared across platforms

`host (java.lang.String)`
The URI of the platform. Must be specified using the full URI (for example, `http://localhost:1234`; `localhost:1234` will not work).

`name (java.lang.String)`
The display name of this Platform instance. Used for identifying different platforms in the user interface.

`aliases (java.lang.String)`
Manage mappings from field names to aliases (use a comma-separated list). All references to the field (via this platform) in results, facets, filters, and query strings are mapped. For example, to refer to a field in the index named `firstnamelastname` as `name` and `countryofresidence` as `country`, use these mappings:
`aliases="firstnamelastname=name,countryofresidence=country"`

`defaultQuery (java.lang.String)`
Default query to use when none is specified.

`defaultFacets (java.lang.String)`
Default facets to request when none are specified. This is a comma separated list.

`spellCheck (java.lang.Boolean)`
Spellcheck the Query term if supported by the Platform.
Default: `true`

`expandQuery (java.lang.Boolean)`
Apply advanced linguistics such as stemming or lemmatization if supported by the Platform.
Default: `true`

`autoCorrect (java.lang.Boolean)`
Whether to auto-correct and resubmit futile queries (queries with zero results).
Default: `true`

`fileTypeField (java.lang.String)`
Set which field contains information about file type (mime type).

`resultIDField (java.lang.String)`
Set which field represents the unique identifier for a given result.

`pageLimit (java.lang.Long)`
A limitation to which page the platform will offset.

Lucidworks Fusion

The Fusion search adapter brokers all communication with the Lucidwork's Fusion platform *(https://lucidworks.com/products/fusion/)* using its REST API to query Solr.

Setup

To add Fusion search connectors to a Maven project, you must add this dependency to your project's `pom.xml` file:

```
<dependency> <groupId>twigkit</groupId> <artifactId>twigkit.fusion</artifactId>
<version>${project.parent.version}</version> </dependency>
```

Usage

Assuming you have added the correct dependencies to your project (see above), the Fusion platform configuration can be accessed by pointing to it in the search:platform tag. In this case the platform is placed in a variable named `platform` using the `var` attribute, and points to a platform configuration in ../conf/platforms/fusion/search.conf:

```
<search:platform var="platform" conf="platforms.fusion.search"></search:platform>
```

platforms/fusion/search.conf

In this example platform configuration, we set some general settings for the platform:

```
name: twigkit.search.fusion.Fusion timeOut: 30000 resultIDField: id highlight: true
defaultQuery: *:* query-profile: my-profile
```

With a Fusion platform, we also need to set some additional parameters in the following files:

services/api/fusion.conf

This file controls which Fusion to use:

```
# Hostname of your Fusion host: myHostname # Port of your Fusion port: 6764 #
Protocol of your Fusion - e.g. http or https protocol: http # Optional: Fusion
Authorization mode - native, serviceaccount, or jwt. # If not configured, this will
default to the first configured option in the order JWT > Service Account > Native
#  Native auth requires no config here, but does require the
twigkit.security.provider.fusion module to be loaded. # auth: native # Service
Account config userName: admin password: password # JWT config # issuer: # key: #
groups: # Optional: If a JWT subject is configured, all requests to Fusion will be
sent as the JWT realm user whose name matches the subject # subject:
```

platforms/fusion/social.conf

This file configures the Lucidworks Appkit Social Module:

```
# For every Fusion App, a corresponding YOUR_FUSION_APP_user_prefs collection is automatically created. # Appkit uses this collection for storing collaboration user-data. # Name of the collection to store user data in collection: Supp-Demo_user_prefs # Name of the Query & Index pipelines to use - _system is the Fusion default # pipeline: _system # Enable writing new Social entities into the platform readOnly: false # Prevent users from searching this platform directly webservice-enabled: false
```

message/service/fusion.conf

This file configures how Signals are sent to Fusion:

```
# Fusion Signals config # Name of the Query profile to send Signals through query-profile: Supp-Demo # The index pipeline to send Signals to # signals-index-pipeline: _signals_ingest # Whether to commit Signals to disk inside Fusion immediately or not # commit: true # Whether to index to Fusion asynchronously or not # async: true
```

Required attributes

`query-profile (java.lang.String)`
The Fusion query profile being accessed. If you have created an app via the Fusion UI called MyAwesomeApp, then you will have a default Query Profile called MyAwesomeApp as well - this is a good starting point.
Applied in: configuration

Optional attributes

`requestHandler (java.lang.String)`
Name of the Request Handler to use. Use this to specify an alternate Request Handler to use for example, the '/dismax' request handler. Note you can override this by specifying a 'custom' attribute on the query.

`requestMethod (java.lang.String)`
Which HTTP request method to use (GET or POST).

`strategy (java.lang.String)`
Whether to cluster multiple hosts (value should be either combine, or round-robin). To enable sharding, set strategy to combine.

`groupField (java.lang.String)`
Setting this enables Solr Result Grouping or Field Collapsing on the given field. This will create a Result for each Group returned. Matching documents for the group will be available as related results.

`timeOut (java.lang.Integer)`
Time out for platform query requests (in milliseconds).
Default: 5000

`ignoreAppliedFilters (java.lang.Boolean)`
If this is set to false, facet filters that have already been applied to the Query are added to the list of filters for the facet.
Default: true

`backwardsCompatible (java.lang.Boolean)`
For Solr 1.x (for example, 1.4) set backwardsCompatible to true.
Default: false

`highlight (java.lang.Boolean)`
Whether to enable hit highlighting (hl parameter) in Solr.
Default: true

Attributes shared across platforms

`aliases (java.lang.String)`
Manage mappings from field names to aliases (use a comma-separated list). All references to the field (via this platform) in results, facets, filters, and query strings are mapped. For example, to refer to a field in the index named `firstnamelastname` as `name` and `countryofresidence` as `country`, use these mappings:
`aliases="firstnamelastname=name,countryofresidence=country"`

`defaultQuery (java.lang.String)`
Default query to use when none is specified. For Solr use `:` to bring back all items (for example, for 'zero term search').

`defaultFacets (java.lang.String)`
Default facets to request when none are specified. This is a comma separated list.

`spellCheck (java.lang.Boolean)`
Spellcheck the Query term if supported by the Platform.
Default: `true`

`expandQuery (java.lang.Boolean)`
Apply advanced linguistics such as stemming or lemmatization if supported by the Platform.
Default: `true`

`autoCorrect (java.lang.Boolean)`
Whether to auto-correct and resubmit futile queries (queries with zero results).
Default: `true`

`fileTypeField (java.lang.String)`
Set which field contains information about file type (mime type).
Applied in: configuration

`resultIDField (java.lang.String)`
Set which field represents the unique identifier for a given result. Should correspond to the <uniqueKey> element in the Solr schema.xml.
Applied in: JSP tag, configuration

`pageLimit (java.lang.Long)`
A limitation to which page the platform will offset.
Applied in: JSP tag, configuration

Hit highlighting

Hit highlighting is enabled by default, and assumes that the unique uniqueKey field in the Solr `schema.xml` file is named id. If your uniqueKey is set to a different field highlighting will not work unless you set the resultIDfield key in the Appkit configuration to the right field.

```
resultIDField = yourUniqueKey
```

Appkit expects that Solr's default highlighting tags of `<em></em>` are used. Ensure that this has not been set to something different using `hl.simple.pre` and `hl.simple.post` settings in the Solr or request handler configurations.

Related articles

- *Fusion Impersonation via a Service Account*
- *Query Terms and Suggestions*

Offline

At times, it is not always possible to have access to your search device. To facilitate this, Appkit offers the ability to record responses and save them to disk when you do have access. Then, you can use those responses to carry on working offline.

Recording responses

By booting your application with the system variable `-Dtwigkit.offline.recording=true`, every search response served by the backend device will be serialized to a JSON file written to disk. By default, the location of the saved response JSON file relative to the application root will be as follows:

```
/offline-data /<platform.id> response_<encoded query url>.json
```

Here, `<platform.id>` is either the configuration name of the platform (for example, `platforms.fusion`) or the platform ID, and `<encoded query url>` is an encoded form of the query URL that was used to generate the response. The default offline root folder, `offline-data`, can be overridden by setting the system variable `-Dtwigkit.offline.recording.path` to a relative or absolute file path.

For example, suppose you submit a query to the `platforms.solr` platform instance, with `-Dtwigkit.offline.recording.path=my-data`. Then the system will save any response serialized as a JSON file to:

```
my-data platforms.solr response_1234567.json
```

If a default query (with no query or filter terms) is run, this response will be written to `response.json` and saved in the appropriate platform folder.

Working offline

The Offline Search platform is a search interface that serves static content by matching search queries with JSON data files read from disk. Each JSON data file contains a serialized Appkit response object, independent of the particular search engine that produced the original search results. See the section above for information on how to capture and store search response data.

To work offline with recorded responses, the first step is to add this dependency to your project's `pom.xml` file:

```
<dependency> <groupId>twigkit</groupId>
<artifactId>twigkit.offline.search</artifactId> <version>${project.parent.version}
</version> </dependency>
```

After this is done, the next step is to update the configuration file of the platform you had been working online with in order to tell it that you are now working offline. The best way to do this is to use the *configuration overlay*. For example, if you previously recorded a number of responses from a Solr device, you could create or update the configuration file, `solr.conf`, in `src/dev/resources/conf/platforms/solr/` to have these entries:

```
name: twigkit.search.offline.OfflineSearch path: file://data-files/platforms.solr
```

Here, `path` defines the location of the offline response JSON files. This can either be a relative path (for example, ./data-files), resolved relative to the directory from which the web server is booted, or an absolute path (for example, /var/data-files).

Pipl

The Pipl search platform adds support to Appkit for the Pipl search engine for detailed people results from a range of sources.

Setup

To add Pipl connectors to a Maven project, you must add this dependency to your project's `pom.xml` file:

```
<dependency> <groupId>twigkit</groupId> <artifactId>twigkit.pipl</artifactId>
<version>${project.parent.version}</version> </dependency>
```

Required attributes

`apiKey (java.lang.String)`
The API Key for programmatic access to Pipl.

Attributes shared across platforms

`host (java.lang.String)`
The URI of the platform. Must be specified using the full URI (for example, `http://localhost:1234`; `localhost:1234` will not work).

`name (java.lang.String)`
The display name of this Platform instance. Used for identifying different platforms in the user interface.

`aliases (java.lang.String)`
Manage mappings from field names to aliases (use a comma-separated list). All references to the field (via this platform) in results, facets, filters, and query strings are mapped. For example, to refer to a field in the index named `firstnamelastname` as `name` and `countryofresidence` as `country`, use these mappings:
`aliases="firstnamelastname=name,countryofresidence=country"`

`defaultQuery (java.lang.String)`
Default query to use when none is specified.

`defaultFacets (java.lang.String)`
Default facets to request when none are specified. This is a comma separated list.

`spellCheck (java.lang.Boolean)`
Spellcheck the Query term if supported by the Platform.
Default: `true`

`expandQuery (java.lang.Boolean)`
Apply advanced linguistics such as stemming or lemmatization if supported by the Platform.
Default: `true`

`autoCorrect (java.lang.Boolean)`
Whether to auto-correct and resubmit futile queries (queries with zero results).
Default: `true`

`fileTypeField (java.lang.String)`
Set which field contains information about file type (mime type).

`resultIDField (java.lang.String)`
Set which field represents the unique identifier for a given result.

`pageLimit (java.lang.Long)`
A limitation to which page the platform will offset.

Shared Platform Attributes

This is a list of shared attributes common to all *search platforms*.

Where applicable, default values are given, for example, the default value for the `spellcheck` attribute is set to `true`. An indication is also given of where these attributes can be applied. For example, some attributes can be applied either in the JSP tag or a configuration file (for example, `gsa.conf`). Other attributes can only be applied in the JSP tag or in the configuration file.

On some platforms, the values following `Applied in:` might not be exactly the same as those shown below. For confirmation, also refer to the list of standard attributes specified on that platform page.

Attributes shared across platforms

`host (java.lang.String)`
The URI of the platform. Must be specified using the full URI (for example, `http://localhost:1234`; `localhost:1234` will not work).

`name (java.lang.String)`
The display name of this Platform instance. Used for identifying different platforms in the user interface.

`aliases (java.lang.String)`
Manage mappings from field names to aliases (use a comma-separated list). All references to the field (via this platform) in results, facets, filters, and query strings are mapped. For example, to refer to a field in the index named `firstnamelastname` as `name` and `countryofresidence` as `country`, use these mappings:
`aliases="firstnamelastname=name,countryofresidence=country"`

`defaultQuery (java.lang.String)`
Default query to use when none is specified.

`defaultFacets (java.lang.String)`
Default facets to request when none are specified. This is a comma separated list.

`spellCheck (java.lang.Boolean)`
Spellcheck the Query term if supported by the Platform.
Default: `true`

`expandQuery (java.lang.Boolean)`
Apply advanced linguistics such as stemming or lemmatization if supported by the Platform.
Default: `true`

`autoCorrect (java.lang.Boolean)`
Whether to auto-correct and resubmit futile queries (queries with zero results).
Default: `true`

`fields (java.lang.String)`
Names of fields to request. This is a comma separated list. Fields set on the query take precedence over fields set here.

`fileTypeField (java.lang.String)`
Set which field contains information about file type (mime type).

`resultIDField (java.lang.String)`
Set which field represents the unique identifier for a given result.

`pageLimit (java.lang.Long)`
A limitation to which page the platform will offset.

`webservice-enabled (java.lang.String)`
Whether the platform web service should be accessible (defaults to 'true'). Set to 'false' if you do not want to expose the web service for the platform to authenticated users. Useful when a workflow platform implements additional security trimming so the underlying platform web service should be switched off to avoid unauthorized access to a wider set of documents.

Twitter

The platform adapter for Twitter allows Appkit applications to search Twitter or retrieve a user's timeline using the Twitter API.

Setup

To add Twitter connectors to a Maven project, you must add this dependency to your project's `pom.xml` file:

```
<dependency> <groupId>twigkit</groupId> <artifactId>twigkit.twitter</artifactId>
<version>${project.parent.version}</version> </dependency>
```

Usage

This example shows how to configure a Twitter platform.

```
name: twigkit.search.twitter.Twitter oAuthConsumerKey: foo oAuthConsumerSecret: foo
oAuthAccessToken: foo oAuthAccessSecret: foo
```

Required attributes

`oAuthConsumerKey (java.lang.String)`
OAuth consumer key for the Twitter API.

`oAuthConsumerSecret (java.lang.String)`
OAuth consumer secret for the Twitter API.

`oAuthAccessToken (java.lang.String)`
OAuth access token for the Twitter API.

`oAuthAccessSecret (java.lang.String)`
OAuth access secret for the Twitter API.

Optional attributes

`userTimeline (java.lang.String)`
User timeline to retrieve.

Attributes shared across platforms

`host (java.lang.String)`
The URI of the platform. Must be specified using the full URI (for example, `http://localhost:1234`; `localhost:1234` will not work).

`name (java.lang.String)`
The display name of this Platform instance. Used for identifying different platforms in the user interface.

`aliases (java.lang.String)`
Manage mappings from field names to aliases (use a comma-separated list). All references to the field (via this platform) in results, facets, filters, and query strings are mapped. For example, to refer to a field in the index named `firstnamelastname` as `name` and `countryofresidence` as `country`, use these mappings:
`aliases="firstnamelastname=name,countryofresidence=country"`

`defaultQuery (java.lang.String)`
Default query to use when none is specified.

`defaultFacets (java.lang.String)`
Default facets to request when none are specified. This is a comma separated list.

`spellCheck (java.lang.Boolean)`
Spellcheck the Query term if supported by the Platform.
Default: `true`

`expandQuery (java.lang.Boolean)`
Apply advanced linguistics such as stemming or lemmatization if supported by the Platform. Default: `true`

`autoCorrect (java.lang.Boolean)`
Whether to auto-correct and resubmit futile queries (queries with zero results).
Default: `true`

`fileTypeField (java.lang.String)`
Set which field contains information about file type (mime type).

`resultIDField (java.lang.String)`
Set which field represents the unique identifier for a given result.

`pageLimit (java.lang.Long)`
A limitation to which page the platform will offset.

Wikipedia

The Wikipedia platform adds support to Appkit for the Wikipedia search engine.

Setup

To add Wikipedia connectors to a Maven project, you must add this dependency to your project's `pom.xml` file:

```
<dependency> <groupId>twigkit</groupId> <artifactId>twigkit.wikipedia</artifactId>
<version>${project.parent.version}</version> </dependency>
```

Usage

This example shows how to configure a Wikipedia search endpoint. You can define the search endpoint through a configuration node (say `conf/platforms/wikipedia.conf`). For this you must include `name: twigkit.search.wikipedia.Wikipedia`, as shown here:

```
name: twigkit.search.wikipedia.Wikipedia defaultQuery: my_default_query
```

Attributes shared across platforms

`host (java.lang.String)`
The URI of the platform. Must be specified using the full URI (for example, `http://localhost:1234`; `localhost:1234` will not work).

`name (java.lang.String)`
The display name of this Platform instance. Used for identifying different platforms in the user interface.

`aliases (java.lang.String)`
Manage mappings from field names to aliases (use a comma-separated list). All references to the field (via this platform) in results, facets, filters, and query strings are mapped. For example, to refer to a field in the index named `firstnamelastname` as `name` and `countryofresidence` as `country`, use these mappings:
`aliases="firstnamelastname=name,countryofresidence=country"`

`defaultQuery (java.lang.String)`
Default query to use when none is specified.

`defaultFacets (java.lang.String)`
Default facets to request when none are specified. This is a comma separated list.

`spellCheck (java.lang.Boolean)`
Spellcheck the Query term if supported by the Platform.
Default: `true`

`expandQuery (java.lang.Boolean)`
Apply advanced linguistics such as stemming or lemmatization if supported by the Platform.
Default: `true`

`autoCorrect (java.lang.Boolean)`
Whether to auto-correct and resubmit futile queries (queries with zero results).
Default: `true`

`fileTypeField (java.lang.String)`
Set which field contains information about file type (mime type).

`resultIDField (java.lang.String)`
Set which field represents the unique identifier for a given result.

`pageLimit (java.lang.Long)`
A limitation to which page the platform will offset.

Wolfram|Alpha

The Wolfram|Alpha platform adds support to Appkit for the Wolfram|Alpha *(http://www.wolframalpha.com/)* computational knowledge engine.

Setup

To add Wolfram|Alpha connectors to a Maven project, you must add this dependency to your project's `pom.xml` file:

```
<dependency> <groupId>twigkit</groupId>
<artifactId>twigkit.wolframalpha</artifactId> <version>${project.parent.version}
</version> </dependency>
```

Usage

Assuming you have added the correct dependencies to your project (see above), the Wolfram Alpha platform configuration can be accessed by pointing to it in the search:platform tag. In this case the platform is placed in a variable named `platform` using the `var` attribute, and points to a platform configuration in ../conf/platforms/wa/wa.conf:

```
<search:platform var="platform" conf="platforms.wa"></search:platform>
```

platforms/solr/solr.conf

In this example platform configuration, we set some general settings for the platform:

```
name: twigkit.search.wolframalpha.WolframAlpha appId: ABCD-EFGH
```

Required attributes

`appId (java.lang.String)`
Application ID API key.

Optional attributes

`async (java.lang.Boolean)`
Whether to request data in 'asynchronous' mode or not. In asynchronous mode, only the definitions of existing properties are returned in the first response, with links to further information or computations.
Default: `false`

Attributes shared across platforms

`host (java.lang.String)`
The URI of the platform. Must be specified using the full URI (for example, `http://localhost:1234`; `localhost:1234` will not work).

`name (java.lang.String)`
The display name of this Platform instance. Used for identifying different platforms in the user interface.

`aliases (java.lang.String)`
Manage mappings from field names to aliases (use a comma-separated list). All references to the field (via this platform) in results, facets, filters, and query strings are mapped. For example, to refer to a field in the index named `firstnamelastname` as `name` and `countryofresidence` as `country`, use these mappings:
`aliases="firstnamelastname=name,countryofresidence=country"`

`defaultQuery (java.lang.String)`
Default query to use when none is specified.

`defaultFacets (java.lang.String)`
Default facets to request when none are specified. This is a comma separated list.

`spellCheck (java.lang.Boolean)`
Spellcheck the Query term if supported by the Platform.
Default: `true`

`expandQuery (java.lang.Boolean)`
Apply advanced linguistics such as stemming or lemmatization if supported by the Platform.
Default: `true`

`autoCorrect (java.lang.Boolean)`
Whether to auto-correct and resubmit futile queries (queries with zero results).
Default: `true`

`fileTypeField (java.lang.String)`
Set which field contains information about file type (mime type).

`resultIDField (java.lang.String)`
Set which field represents the unique identifier for a given result.

`pageLimit (java.lang.Long)`
A limitation to which page the platform will offset.

YouTube

The YouTube platform lets Appkit pull results from YouTube, which is often useful for presenting 'related videos'. The search can be scoped to a particular YouTube Channel if required.

Setup

To add YouTube connectors to a Maven project, you must add this dependency to your project's `pom.xml` file:

```
<dependency> <groupId>twigkit</groupId> <artifactId>twigkit.youtube</artifactId>
<version>${project.parent.version}</version> </dependency>
```

Usage

This example shows how to configure a Yotube search endpoint. You can define the search endpoint through a configuration node (say `conf/platforms/youtube.conf`). For this you must include `name: twigkit.search.youtube.YouTube`, as shown here:

```
name: twigkit.web.youtube.YouTube apiKey: foo channelId: bar
```

Required attributes

`apiKey (java.lang.String)`
API Key for the application to access YouTube.

Optional attributes

`channelId (java.lang.String)`
The ID of the channel to restrict the search to.

Attributes shared across platforms

`host (java.lang.String)`
The URI of the platform. Must be specified using the full URI (for example, `http://localhost:1234`; `localhost:1234` will not work).

`name (java.lang.String)`
The display name of this Platform instance. Used for identifying different platforms in the user interface.

`aliases (java.lang.String)`
Manage mappings from field names to aliases (use a comma-separated list). All references to the field (via this platform) in results, facets, filters, and query strings are mapped. For example, to refer to a field in the index named `firstnamelastname` as `name` and `countryofresidence` as `country`, use these mappings:
`aliases="firstnamelastname=name,countryofresidence=country"`

`defaultQuery (java.lang.String)`
Default query to use when none is specified.

`defaultFacets (java.lang.String)`
Default facets to request when none are specified. This is a comma separated list.

`spellCheck (java.lang.Boolean)`
Spellcheck the Query term if supported by the Platform.
Default: `true`

`expandQuery (java.lang.Boolean)`
Apply advanced linguistics such as stemming or lemmatization if supported by the Platform.
Default: `true`

`autoCorrect (java.lang.Boolean)`
Whether to auto-correct and resubmit futile queries (queries with zero results).
Default: `true`

`fileTypeField (java.lang.String)`
Set which field contains information about file type (mime type).

`resultIDField (java.lang.String)`
Set which field represents the unique identifier for a given result.

`pageLimit (java.lang.Long)`
A limitation to which page the platform will offset.

Filters

Filters are AngularJS based filters that can be implemented in the markup by piping the filter name to a data object, which transforms the object. See *Filter Syntax* for more information.

Filter Syntax

This article describes the syntax of a single Appkit search filter to include in an HTTP request.

```
<filter-expression> ::= <field-expression> <group-expression> <value-expression
<display-expression> <field-expression> ::= <exclusion-expression> <field-optional-
expression> <field-name> <exclusion-expression> ::= "-" | "" <field-optional-
expression> ::= "~" | "" <group-expression> ::= "(" <group-number> ")" <group-
number> ::= <digit> | <digit> <group-number> <field-name> ::= <field-name-prefix>
<field-name-suffix> <field-name-prefix> ::= <letter> | "_" <field-name-suffix> ::=
<field-symbol> | <field-symbol> <field-name-suffix> <field-symbol> ::= <letter> |
<digit> | "_" | "-" | "." | ":" | " " | "" <value-expression> ::= <value-bracket-
left> <filter-value> <value-bracket-right> <value-bracket-left> ::= "[" | "("
<value-bracket-right> ::= "]" | ")" <filter-value> ::= <value> | <range-value>
<value> ::= <string-expression> | <numeric-value> | <date-value> <string-
expression> ::= <partial-match> <string-value> <partial-match> <string-value> ::=
'"' <text> '"' | "'" <text> "'" <partial-match> ::= '' | '*' <numeric-value> ::=
<floating-point> | <integer> <number-sequence> ::= <digit> | <number-sequence>
<digit> <integer> ::= "0" | <positive-digit> <number-sequence> <floating-point> ::=
<integer> | <integer> "." <number-sequence> <date-value> ::= <four-digits> "-"
<two-digits> "-" <two-digits> "'T'" <two-digits> ":" <two-digits> ":" <two-digits>
<two-digits> ::= <digit> <digit> <four-digits> ::= <two-digits> <two-digits>
<range-value> ::= <value> "," <value> | "," <value> | <value> "," <display-
expression-opt> ::= "" | "[" <display-expression> "]" <display-expression> ::= "["
'"' <display-value> '"' "]" | "[" "'" <display-value> "'" "]"
```

Where:

```
<letter> ::= "A" | "B" | "C" | "D" | "E" | "F" | "G" | "H" | "I" | "J" | "K" | "L"
| "M" | "N" | "O" | "P" | "Q" | "R" | "S" | "T" | "U" | "V" | "W" | "X" | "Y" | "Z"
| "a" | "b" | "c" | "d" | "e" | "f" | "g" | "h" | "i" | "j" | "k" | "l" | "m" | "n"
| "o" | "p" | "q" | "r" | "s" | "t" | "u" | "v" | "w" | "x" | "y" | "z" ;
<positive-digit> ::= "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9" <digit>
::= "0" | <positive-digit>
```

Here:

- `<group-number>` is an integer value denoting a grouping of filter constraints
- `<field-name>` is the name of the field to apply the constraint to. A valid field name is one that matches the regular expression `[A-Za-z_][A-Za-z_0-9\-\.: ]`
- `<display-value>` is an optional label to attach to the filter
- `<filter-value>` is the actual constraint to be applied to the field named <field-name>

actual

Description

Renders fields actual value

Usage

In HTML Template Binding

```
{{ actual_expression | actual }}
```

In JavaScript

```
$filter('actual')(string)
```

Parameters

Param	Type	Details
string	string	The field to get the actual value from

actualValue

Description

Renders fields actual value

Usage

In HTML Template Binding

```
{{ actualValue_expression | actualValue }}
```

In JavaScript

```
$filter('actualValue')(string)
```

Parameters

Param	Type	Details
string	string	The field to get the actual value from

dateFormat

Description

Formats `date` to a string based on the requested `format`.

`format` string can be composed of the following elements:

- `'yyyy'`: 4 digit representation of year (e.g. AD 1 => 0001, AD 2010 => 2010)
- `'yy'`: 2 digit representation of year, padded (00-99). (e.g. AD 2001 => 01, AD 2010 => 10)
- `'y'`: 1 digit representation of year, e.g. (AD 1 => 1, AD 199 => 199)
- `'MMMM'`: Month in year (January-December)
- `'MMM'`: Month in year (Jan-Dec)
- `'MM'`: Month in year, padded (01-12)
- `'M'`: Month in year (1-12)
- `'LLLL'`: Stand-alone month in year (January-December)
- `'dd'`: Day in month, padded (01-31)
- `'d'`: Day in month (1-31)
- `'EEEE'`: Day in Week,(Sunday-Saturday)
- `'EEE'`: Day in Week, (Sun-Sat)
- `'HH'`: Hour in day, padded (00-23)
- `'H'`: Hour in day (0-23)
- `'hh'`: Hour in AM/PM, padded (01-12)
- `'h'`: Hour in AM/PM, (1-12)
- `'mm'`: Minute in hour, padded (00-59)
- `'m'`: Minute in hour (0-59)
- `'ss'`: Second in minute, padded (00-59)
- `'s'`: Second in minute (0-59)
- `'sss'`: Millisecond in second, padded (000-999)
- `'a'`: AM/PM marker
- `'Z'`: 4 digit (+sign) representation of the timezone offset (-1200-+1200)
- `'ww'`: Week of year, padded (00-53). Week 01 is the week with the first Thursday of the year
- `'w'`: Week of year (0-53). Week 1 is the week with the first Thursday of the year
- `'G'`, `'GG'`, `'GGG'`: The abbreviated form of the era string (e.g. 'AD')
- `'GGGG'`: The long form of the era string (e.g. 'Anno Domini')
- `'medium'`: equivalent to `'MMM d, y h:mm:ss a'`
- `'short'`: equivalent to `'M/d/yy h:mm a'`
- `'fullDate'`: equivalent to `'EEEE, MMMM d, y'`
- `'longDate'`: equivalent to `'MMMM d, y'`
- `'mediumDate'`: equivalent to `'MMM d, y'`
- `'shortDate'`: equivalent to `'M/d/yy'`
- `'mediumTime'`: equivalent to `'h:mm:ss a'`
- `'shortTime'`: equivalent to `'h:mm a'`
- `'relative'`: displays the time since/until the date

`format` string can contain literal values. These need to be escaped by surrounding with single quotes (e.g. `"h 'in the morning'"`). In order to output a single quote, escape it - i.e., two single quotes in a sequence (e.g. `"h 'o''clock'"`).

Usage

In HTML Template Binding

```
{{ dateFormat_expression | dateFormat:format[:timezone] }}
```

In JavaScript

```
$filter('dateFormat')(date, format[, timezone])
```

Parameters

Param	Type	Details
date	Datenumberstring	The date value to format.
format	string	The format to render the date value in.
timezone *(optional)*	string	Timezone to be used for formatting. It understands UTC/GMT and the continental US time zone abbreviations, but for general use, use a time zone offset, for example, '+0430' (4 hours, 30 minutes east of the Greenwich meridian) If not specified, the timezone of the browser will be used.

deepFilter

Description

Returns a subset of the given array, containing all the elements matching the given object filter. A filter is specified by a selector identifying a property of the objects in the array, and a value to match that property against. Matches are left-partial meaning that a match against "ca" will match both "car" and "cat".

Usage

In HTML Template Binding

```
{{ deepFilter_expression | deepFilter:path:target }}
```

In JavaScript

```
$filter('deepFilter')(array, path, target)
```

Parameters

Param	Type	Details
array	array	Array of objects to filter on.
path	string	Path of the property to search for the target value
target	string	The value you are looking to match against.

display

Description

Renders fields actual value

Usage

In HTML Template Binding

```
{{ display_expression | display }}
```

In JavaScript

```
$filter('display')(string)
```

Parameters

Param	Type	Details
string	string	The field to get the actual value from

displayValue

Description

Renders fields display value

Usage

In HTML Template Binding

```
{{ displayValue_expression | displayValue }}
```

In JavaScript

```
$filter('displayValue')(string)
```

Parameters

Param	Type	Details
string	string	The field to get the display value from

encodeURIComponent

Description

Renders fields actual value

Usage

In HTML Template Binding

```
{{ encodeURIComponent_expression | encodeURIComponent }}
```

In JavaScript

```
$filter('encodeURIComponent')(string)
```

Parameters

Param	Type	Details
string	string	The value to encode

facet

Description

Returns a result from the response

Usage

In HTML Template Binding

```
{{ facet_expression | facet:facetName }}
```

In JavaScript

```
$filter('facet')(response, facetName)
```

Parameters

Param	Type	Details
response	object	The response object to get the facet from
facetName	string	The facet name

fact

Description

Returns the first fact of the provided fact name from the response. If no fact is found it will return null.

Usage

In HTML Template Binding

```
{{ fact_expression | fact:fact-name }}
```

In JavaScript

```
$filter('fact')(response, fact-name)
```

Parameters

Param	Type	Details
response	object	The response object to get the fact from
fact-name	string	The fact name

field

Description

Returns a result from the response

Usage

In HTML Template Binding

```
{{ field_expression | field[:field] }}
```

In JavaScript

```
$filter('field')(result[, field])
```

Parameters

Param	Type	Details
result	string	The result object to get the field from.
field *(optional)*	number	The name of the field to retrieve

filterCount

Description

Returns a result from the response

Usage

In HTML Template Binding

```
{{ filterCount_expression | filterCount[:value] }}
```

In JavaScript

```
$filter('filterCount')(facet[, value])
```

Parameters

Param	Type	Details
facet	string	The facet object to get the filter from
value *(optional)*	number	The value of the filter to get

filterValue

Description

Returns the facet filter that has the requested value

Usage

In HTML Template Binding

```
{{ filterValue_expression | filterValue[:value] }}
```

In JavaScript

```
$filter('filterValue')(facet[, value])
```

Parameters

Param	Type	Details
facet	string	The facet object to get the filter from
value *(optional)*	number	The value of the filter to get

formatAsCssName

Description

Renders a string in a friendly name for CSS.

Usage

In HTML Template Binding

```
{{ formatAsCssName_expression | formatAsCssName }}
```

In JavaScript

```
$filter('formatAsCssName')(string)
```

Parameters

Param	Type	Details
string	string	The string to format.

getUrlForQuery

Description

Returns the query parameters for a query object

Usage

In HTML Template Binding

```
{{ getUrlForQuery_expression | getUrlForQuery }}
```

In JavaScript

```
$filter('getUrlForQuery')(query)
```

Parameters

Param	Type	Details
query	Object	The query object

hasFilter

Description

Returns the first filter object from the query or facet that matches the name and optionally the value also

Usage

In HTML Template Binding

```
{{ hasFilter_expression | hasFilter:field[:value] }}
```

In JavaScript

```
$filter('hasFilter')(query, field[, value])
```

Parameters

Param	Type	Details
query	string	The query object
field	number	The name of the filter
value *(optional)*	number	The value of the filter

htmlToPlainText

Description

Strip HTML from a string.

Usage

In HTML Template Binding

```
{{ htmlToPlainText_expression | htmlToPlainText }}
```

In JavaScript

```
$filter('htmlToPlainText')(string)
```

Parameters

Param	Type	Details
string	string	The value to strip the html from.

maxCharacters

Description

Renders fields actual value

Usage

In HTML Template Binding

```
{{ maxCharacters_expression | maxCharacters:maxCharacters }}
```

In JavaScript

```
$filter('maxCharacters')(string, maxCharacters)
```

Parameters

Param	Type	Details
string	string	The field to get the actual value from
maxCharacters	number	Number of characters to display

numberFormat

Description

Renders a number in the format specified.

Usage

In HTML Template Binding

```
{{ numberFormat_expression | numberFormat[:format] }}
```

In JavaScript

```
$filter('numberFormat')(number[, format])
```

Parameters

Param	Type	Details
number	number	The number to format.
format *(optional)*	string	string for the numbers used using the Java DecimalFormat pattern. Default: '#,##0.###'

orderByValue

Description

Orders values from facet or results

Usage

In HTML Template Binding

```
{{ orderByValue_expression | orderByValue }}
```

In JavaScript

```
$filter('orderByValue')([desc])
```

Parameters

Param	Type	Details
desc *(optional)*	boolean	Whether it should be descending or not.

placeholder

Description

Renders a placeholder while a field is null.

Usage

In HTML Template Binding

```
{{ placeholder_expression | placeholder:placeholder }}
```

In JavaScript

```
$filter('placeholder')(input, placeholder)
```

Parameters

Param	Type	Details
input	string	The input to check if it is null
placeholder	string	The value to display while input is null

property

Description

Get value from an array based on the value of a property.

Usage

In HTML Template Binding

```
{{ property_expression | property:value:returnProperty }}
```

In JavaScript

```
$filter('property')(property, value, returnProperty)
```

Parameters

Param	Type	Details
property	string	The name of the property to filter on.
value	string	The value of the property to filter on.
returnProperty	string	The property to return.

pyramidSort

Description

Sorts values in a pyramid fashion

Usage

In HTML Template Binding

```
{{ pyramidSort_expression | pyramidSort:field }}
```

In JavaScript

```
$filter('pyramidSort')(items, field)
```

Parameters

Param	Type	Details
items	object	Array of values
field	string	The field to sort on.

rangeDateFormat

Deprecated API

Description

Range date format filter has been deprecated, please use the *Date Format filter*.

Usage

In HTML Template Binding

```
{{ rangeDateFormat_expression | rangeDateFormat }}
```

In JavaScript

```
$filter('rangeDateFormat')()
```

result

Description

Returns a result from the response

Usage

In HTML Template Binding

```
{{ result_expression | result[:index] }}
```

In JavaScript

```
$filter('result')(response[, index])
```

Parameters

Param	Type	Details
response	string	The response object to get the result from.
index *(optional)*	number	The index of the result to retrieve, index starts at 0. (Defaults to 0).

roundedNumber

Description

Renders a number rounded, eg. 1234 = 1.2k

Usage

In HTML Template Binding

```
{{ roundedNumber_expression | roundedNumber }}
```

In JavaScript

```
$filter('roundedNumber')(value)
```

Parameters

Param	Type	Details
value	number	The number to round.

translate

Description

Translate a term using the Translation Service

Usage

In HTML Template Binding

```
{{ translate_expression | translate[:bundle] }}
```

In JavaScript

```
$filter('translate')(term[, bundle])
```

Parameters

Param	Type	Details
term	string	The term to translate
bundle *(optional)*	string	The translation bundle to use

trusted

Description

Returns a trusted url. This is required if trying to use an angular expression in the media idea src attribute.

Usage

In HTML Template Binding

```
{{ trusted_expression | trusted }}
```

In JavaScript

```
$filter('trusted')(url)
```

Parameters

Param	Type	Details
url	string	The url to return.

Installation

This section contains various articles on the installation, system requirements, and supported environments of App Studio.

Appkit is most commonly run as a lightweight Java web application that can be deployed in any Java Servlet Container *(http://en.wikipedia.org/wiki/Java_Servlet#Container_servers)* (application server) such as Tomcat *(http://tomcat.apache.org/)*, Glassfish *(http://glassfish.java.net/)*, Jetty *(http://www.eclipse.org/jetty/)* depending on your preferences or IT standards. The Appkit application is designed to be stateless *(http://en.wikipedia.org/wiki/Stateless_protocol)*, making deployment and scaling (for example, with load balancers *(http://en.wikipedia.org/wiki/Load_balancing_(computing))* a straightforward exercise.

Requirements

App Studio is a standalone application with the following requirements:

- Oracle Java JDK *(https://www.oracle.com/technetwork/java/javase/downloads/index.html)* 1.8 or later
- A running instance of *Fusion*, with one or more collections of indexed data

Installation

1. *Download App Studio Enterprise*.
2. Copy the App Studio Enterprise zip file to any convenient path.
3. Unzip the file, and follow the instructions provided in the `README.md` file.

> Starting in Fusion 5.0, App Studio is no longer included in the Fusion UI. However, you can use App Studio Enterprise (ASE) to create apps in Fusion 5 *and* Fusion 4. See *How to Deploy App Studio Enterprise* for instructions.

Configuring the application server

For development and testing purposes you can deploy the Appkit web application in any modern servlet container and it will run and serve queries as long as the computer has network access to the underlying search engine. This is a non-exhaustive list of servers that have been used to run Appkit applications:

- Tomcat 6+
- JBoss 8+
- Jetty 6+
- GlassFish 4+

Java runtime environment

Appkit requires Java JDK 8 to be installed on the application server. Java EE is not needed.

Memory allocation

You must make sure that the servlet container is configured with access to an adequate amount of memory and heap space. The standard Appkit application does not have specific requirements but specific modules (for example, the *Social Module* have a larger footprint. See the *Troubleshooting page* for more information.

Typical production setup

Your production setup will be a matter of preference because Appkit imposes no specific requirements in terms of topology or infrastructure. Some organisations choose to front the application server (servlet container) with either Apache or another webserver. In a public-facing scenario, this can yield some gains when serving static content and creates a buffer between the outside world and the execution environment.

Using Windows Integrated Authentication for internal applications

If you would like to take advantage of Integrated Windows Authentication *(http://en.wikipedia.org/wiki/Integrated_Windows_Authentication)* for single sign on *(http://en.wikipedia.org/wiki/Single_sign-on)* in your search application, the typical pattern is to front the servlet container (for example, Tomcat) with Microsoft Internet Information Services *(http://www.iis.net/)*. In this pattern IIS handles the authentication with Active Directory and then forwards the request to Tomcat where it is picked up by Appkit and used by our *Security Module* for user information, authorization or personalization.

Related articles

- *Building from Source*
- *Deploying a WAR File*

Open Source Libraries

Below is a list of Open Source Libraries that Appkit uses, along with license details.

Open Source Library	License Details
SLF4J	Logging abstraction framework. MIT License.
Log4j (optional)	Logging framework. Apache 2.0 License.
Commons-IO	Utilities. Apache 2.0 License.
Apache Lucene (optional)	Highlighting. Apache 2.0 License.
WRO4j (optional)	CSS and JavaScript compiler. Apache 2.0 License.
URLRewriteFilter (optional)	URL routing. BSD License.
CORS Filter	Cross-origin scripting capabilities. Apache 2.0 License.
Sitebricks (optional)	Used to provide configuration to Sitebricks. Apache 2.0 License.
Hibernate	Persistence layer integration. Apache 2.0 License.
Spring Security (optional)	Authentication and authorization provider abstraction. Apache 2.0 License.
JodaTime	Date calculations. Apache 2.0 License.
Jasypt	Encryption/decryption. Apache 2.0 License.
Google Guice	Dependency injection. Apache 2.0 License.
Jersey (optional)	Web services. CDDL 1.1 and GPL 2 with class path exception licenses.
OWASP ESAPI	Security and vulnerability layer. BSD License.
Bouncy Castle License	Cryptography. MIT License.
Apache Derby (optional)	Database. Apache 2.0 License.
Markdownj (optional)	Markdown parsing. Custom license, allows redistribution and use through attribution.
Rome (optional)	RSS parser and generator. Apache 2.0 License.
Jackson	JSON parsing and generation. Apache 2.0 and LGPL licenses.
Quartz Scheduler (optional)	Event scheduling for saved searches and alerting. Apache 2.0 License.
Plexus	Application building. Apache 2.0 License.
JUnit	Unit testing - not included in deliverable. IBM Common Public License 0.5.
Atunit	Unit testing - not included in deliverable. Apache 2.0 License.

Open Source Library	License Details
EasyMock	Unit testing - not included in deliverable. Apache 2.0 License.
ImageIO-Ext (optional)	Rich media transcoding. LGBP 2.1 and BSD licenses.

Software Requirements

This software is required to use App Studio and Appkit to develop, build, and deploy search apps.

Software	Version and Notes
Appkit	A version of Appkit is included in each App Studio release. Appkit is also released independently. For information about upgrading an app to use a later version of Appkit, see link:/how-to/791/upgrade-appkit-in-existing-apps [Upgrade Appkit in existing apps].
Oracle Java 8 JDK or later	Download *(https://www.oracle.com/technetwork/java/javase/downloads/index.html)* and install the Oracle Java SE JDK 8 or newer for Windows *(http://www.oracle.com/technetwork/java/javase/documentation/install-windows-152927.html)* or Linux *(http://www.oracle.com/technetwork/java/javase/install-linux-self-extracting-138783.html)* (it comes pre-installed on Macs).
Node.js 8 or later	Download *(https://nodejs.org/en/download/)* and install Node.js 8 or later.
npm	The `npm` package manager is included with Node.js.

Supported Devices and Browsers

What browsers can be supported? For example IE 11 or 9?
Any modern browser will be perfectly fine. For Internet Explorer, we recommend at least version 11. Appkit uses progressive enhancements so for older browsers the application will still work but in a feature limited fashion.

How does AppKit support responsive design?
All standard Appkit templates are responsively designed and will work correctly on tablets and mobiles with full feature parity.

What is the process we must follow to accomplish responsive design?
As long as the existing templates are not compromised, responsive design will work out of the box. The user interface being adapted is a customization of a fully responsive template.

Licensing

This article contains basic licensing details.

App Studio is licensed *with Fusion*.

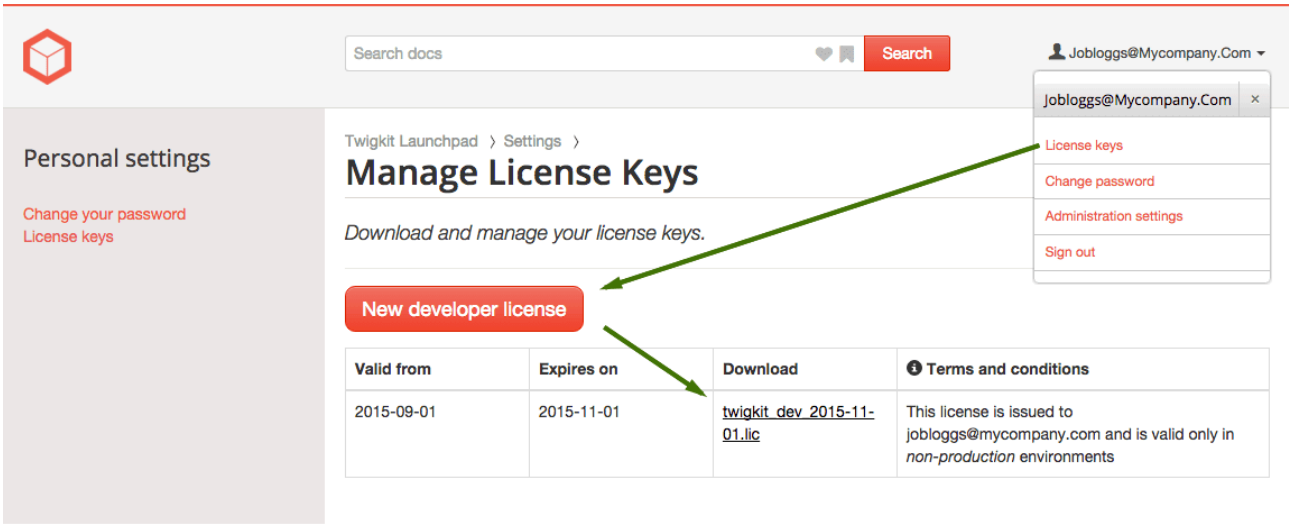
What is an Appkit license file?

To run an Appkit application, you need an Appkit license file. This is a small encrypted text file that an Appkit application reads on startup. The license file enables the individual Appkit *modules* and *search platforms* that you need for your application.

Types of licenses

Development licenses

To run an app in **development**, you will need a developer license from us. To generate a development license, go to the Manage License Keys page *(https://dev.twigkit.net/settings/keys/)* (accessed via the drop down menu by your logged-in username) and click 'new developer license':



These are usually time-limited, enable all Appkit features, and not tied to any given hostname, so you can run them and develop locally, and try out all Appkit features.

Production licenses

For **production**, two tiers of licensing are available: *per host* and *enterprise*.

- **Per-host.** The easiest way to get started with Appkit is through our low-cost, host-based licenses. As the name suggests, these licenses are bound to a single machine hostname. The license is purchased to cover an agreed period (usually 1 year). They are bound to specific modules.
- **Enterprise.** Lucidworks also provides enterprise-wide licenses for customers wanting unlimited usage of Appkit throughout their entire organization. These annual licenses include access to source code, greater involvement in our development roadmap, and premium developer support.

For more information on Per-host and Enterprise licensing, contact Lucidworks *(https://lucidworks.com/company/contact/)*.

How do I add a license?

License files should be named `app-studio.lic` and stored where the application can read them.

You can specify the location of the license file in the application's configuration tree - specifically, in `src/main/resources/conf/twigkit.conf` you will find this (default) entry

```
license-file: /app-studio.lic
```

This entry can either be *classpath-relative* (as given above) or given as an *absolute file path* (with a `"file://"` `file:///` prefix, for example, `file:///var/opt/lics/app-studio.lic`).

When the license location is given as a relative path, the application tries to look up the license file on the runtime class path. So you can place the license in for example, the src/main/resources folder in your application, which will work. If no license file is found on the class path, it looks in the home directory of the user that spun up the JVM. The latter is useful for development, by having a single license on the dev workstation.

License errors and problems

If you are seeing license errors or exceptions (like 'twigkit.license.LicenseException: null at twigkit.license.PLM.isValid'), it could be that you have either mis-configured your license (for example, it is not named `app-studio.lic`), or it is being used on the wrong host or not visible by the application.

Contact Lucidworks Support *(https://support.lucidworks.com/hc/en-us/requests/new)* and provide your hostname, license file, and full details of the error in the Appkit logs, and we will be able to advise further.

Styles

This article lists the valid values for the `styling` attribute supported by many App Studio markup tags.

Global styles

These styles can be applied to any element.

Flex utilities

- `stretch-content`: Uses flex box to stretch the child content
  See the *`grid`* and *`panel`* examples in the gallery.
- `stretch-row`: Same as stretch-content but with auto height for use with rows
- `center-content`: Center child content vertically and horizontally
  See the *`grid`* and *`panel`* examples in the gallery.
- `flex-nowrap`: Prevent flex wrapping
- `flex-grow`: Flex grow an element

Width utilities

- `width-1`: 100%
- `width-2-3`: 66%
- `width-1-2`: 50%
- `width-1-3`: 33%
- `width-1-4`: 25%

Display utilities

- `inline-center`
- `block`
- `inline`
- `inline-block`
- `pull-left`: float left
  See the *`button`* example in the gallery.
- `pull-right`: float right
  See the *`button`* example in the gallery.
- `floated left`: float element left
- `floated right`: float element right
- `border`: Add 1px border to element
- `border right`: Add 1px right border to element
- `border left`: Add 1px left border to element
- `border bottom`: Add 1px bottom border to element
- `border top`: Add 1px top border to element
- `centercontent`: adds padding that pushes in from page edges.
- `centercontent-inner`: adds borders and box-shadows etc. - should be used inside `centercontent`
- `centercontent-inner-shadow`: Same as centercontent but with a drop shadow
- `fade`: Fades out an element, add the class in to fade back in
- `bg-stretch`: Stretch a background image to the full element height and width

Headers

- `header header-fixed`: position fixed header
- `header header-shadow`: header with drop shadow
- `header header-gradient`: header with gradient
- `header header-border`

Sizes

- `xs`: Make the font-size the smallest - also works for making most elements smaller
- `sm`
- `md`
- `lg`
- `xl`
- `break-text-all`: Make text wrap mid word

- `break-text-normal`: Text wraps at best fit

Border radius

- `border-radius`: Adds rounded corners
- `border-radius-top-left`: Adds rounder corner to top left
- `border-radius-top-right`
- `border-radius-bottom-left`
- `border-radius-bottom-right`

Typography

- `text-center`
- `text-right`
- `text-left`
- `capitalize`
- `uppercase`
- `lowercase`

Colors

See these *colors* in the gallery.

- `bg-lgrey`: Light grey background
- `bg-mgrey`
- `bg-dgrey`
- `bg-charcoal`
- `bg-black`
- `bg-white`
- `text-white`: White text
- `bg-lbgrey`
- `bg-mbgrey`
- `bg-dbgrey`
- `bg-blue`
- `bg-sblue`
- `bg-yellow`
- `bg-orange`
- `bg-red`
- `bg-pink`
- `bg-purple`
- `bg-green`
- `bg-primary`
- `bg-primary-light`
- `bg-primary-dark`
- `bg-primary-contrast`
- `text-primary`
- `text-primary-contrast`
- `bg-secondary`
- `bg-secondary-light`
- `bg-secondary-dark`
- `bg-secondary-contrast`
- `text-secondary`
- `text-secondary-light`
- `text-secondary-dark`
- `text-secondary-contrast`
- `bg-tertiary`
- `bg-tertiary-light`
- `bg-tertiary-dark`
- `bg-tertiary-contrast`
- `text-tertiary`
- `text-tertiary-light`
- `text-tertiary-dark`
- `text-tertiary-contrast`
- `bg-raspberry`

- `bg-raspberry-light`
- `bg-raspberry-dark`
- `text-raspberry`
- `text-raspberry-light`
- `text-raspberry-dark`
- `bg-strawberry`
- `bg-strawberry-light`
- `bg-strawberry-dark`
- `text-strawberry`
- `text-strawberry-light`
- `text-strawberry-dark`
- `bg-carrot`
- `bg-carrot-light`
- `bg-carrot-dark`
- `text-carrot`
- `text-carrot-light`
- `text-carrot-dark`
- `bg-pumpkin`
- `bg-pumpkin-light`
- `bg-pumpkin-dark`
- `text-pumpkin`
- `text-pumpkin-light`
- `text-pumpkin-dark`
- `bg-sunflower-light`
- `bg-sunflower`
- `bg-sunflower-dark`
- `text-sunflower`
- `text-sunflower-light`
- `text-sunflower-dark`
- `bg-leaf`
- `bg-leaf-light`
- `bg-leaf-dark`
- `text-leaf`
- `text-leaf-light`
- `text-leaf-dark`
- `bg-grass`
- `bg-grass-light`
- `bg-grass-dark`
- `text-grass`
- `text-grass-light`
- `text-grass-dark`
- `bg-emerald`
- `bg-emerald-light`
- `bg-emerald-dark`
- `text-emerald`
- `text-emerald-light`
- `text-emerald-dark`
- `bg-mint`
- `bg-mint-light`
- `bg-mint-dark`
- `text-mint`
- `text-mint-light`
- `text-mint-dark`
- `bg-aqua`
- `bg-aqua-light`
- `bg-aqua-dark`
- `text-aqua`
- `text-aqua-light`
- `text-aqua-dark`
- `bg-sky`
- `bg-sky-light`
- `bg-sky-dark`
- `text-sky`

- `text-sky-light`
- `text-sky-dark`
- `bg-royal`
- `bg-royal-light`
- `bg-royal-dark`
- `text-royal`
- `text-royal-light`
- `text-royal-dark`
- `bg-lavender`
- `bg-lavender-light`
- `bg-lavender-dark`
- `text-lavender`
- `text-lavender-light`
- `text-lavender-dark`
- `bg-pink`
- `bg-pink-light`
- `bg-pink-dark`
- `text-pink`
- `text-pink-light`
- `text-pink-dark`
- `bg-paper`
- `bg-paper-light`
- `bg-paper-dark`
- `text-paper`
- `text-paper-light`
- `text-paper-dark`
- `bg-concrete`
- `bg-concrete-light`
- `bg-concrete-dark`
- `text-concrete`
- `text-concrete-light`
- `text-concrete-dark`
- `bg-asphalt`
- `bg-asphalt-light`
- `bg-asphalt-dark`
- `text-asphalt`
- `text-asphalt-light`
- `text-asphalt-dark`
- `bg-duckegg`
- `bg-duckegg-light`
- `bg-duckegg-dark`
- `text-duckegg`
- `text-duckegg-light`
- `text-duckegg-dark`
- `bg-depth-dblue`
- `text-depth-dblue`
- `bg-depth-lblue`
- `text-depth-lblue`

Switches

Usage:

```
<input type="checkbox" styling="checkbox-switch"> <label>Switch Label</label>
```

- `checkbox-switch`: Default switch
  Add styles to label:

```
<input type="checkbox" styling="checkbox-switch"> <label styling="square">Switch Label</label>
```

- `square`
- `flat`

Switch colors

Also added to the label * `positive` * `negative` * `warning` * `info` * `secondary` * `tertiary`

Right-hand-side panel

Adds a wikipedia-esque smart panel * `rhspanel` * `rhspanel-shadow`

Tooltips

- `tooltip`: Adds a tooltip using the aria-label attribute for the tooltip content
- `tooltip-top-left`: Position tooltip
- `tooltip-top-right`
- `tooltip-top`
- `tooltip-bottom-left`
- `tooltip-bottom-right`
- `tooltip-bottom`
- `tooltip-left`
- `tooltip-right`
- `tooltip-top`
- `tooltip-small`
- `tooltip-medium`
- `tooltip-large`
- `tooltip-error`
- `tooltip-warning`
- `tooltip-info`
- `tooltip-success`
- `tooltip-always`: The tooltip is always shown
- `tooltip-rounded`: Rounded corners
- `tooltip-no-animate`
- `tooltip-bounce`

Hero banner

This style is useful for landing pages.

- `hero-banner`

Label

- `label`

Buttons

- `button`: Default button
- `button-hover`: Hover state always active
- `button-secondary`: Secondary color
- `button-green`
- `button-blue`
- `button-white`
- `button-disabled`
- `button-hidden`
- `block`: Full width

Forms

- `checkbox`
- `radio`
- `input-rounded`: Applies only to inputs
- `group`: Form input group
- `form-stacked`: Stacked form - should be applied to the form
- `form-aligned`
- `labelled-input`: Labelled input e.g. an input with a dollar sign attached as a prefix
- `labelled-input label-right`: Labelled input e.g. an input with a percentage sign attached as a suffix
- `styled-checkbox`: CSS checkbox - should be applied to a checkbox input with a label as the next sibling

Lists

These styles can be applied to ordered (`ol`) or unordered (`ul`) HTML elements.

- `striped`
- `lined`

Grid utilities

- `hide-xs`: Hide on XS size screens
- `show-xs`: Show on XS size screens
- `show-sm`
- `show-md`
- `show-lg`
- `show-xl`
- `show-xs`
- `hide-xs`
- `hide-sm`
- `show-sm`
- `hide-md`
- `show-md`
- `hide-lg`
- `show-lg`
- `hide-xl`
- `show-xl`

Component styles

These styles can be applied to specific tags.

Field styles

Styles for the `search:field` tag:

- `title`
- `url`
- `label-left`
- `label-left-always`
- `label-inline`
- `label-above`
- `description`
- `image-top`

Image

Styles for the `media:image` tag:

- `square-crop`
- `circle-crop`

Facet

This style can be applied to the `search:facet` tag:

- `switches`: Render switches instead of checkboxes.
  Unlike the default checkboxes, only one switch can be selected at a time.

Facet list

Styles for `search:facet-list` tag:

- `shadow`
- `shadow-far`
- `recessed`
- `step`
- `simple`
- `wrappedheader`
- `hidemetadata`
- `indented`

Search tabs and nav bars

These styles are for the `search:tabs` tag.

- `navbar`: Navbar style
- `navbar navbar-shadow`
- `navbar navbar-centered`
- `navbar navbar-tabs`: Tab style of navbar
- `navbar navbar-light`: Light version
- `tabs`: Tabs style

Comment list

Styles for the `collaborate:comment-list` tag:

- `hide-avatars`: Hides the comment avatars

Pagination

Styles for `search:pagination` tag.

- `pagination-wrapper-left`: Align left
- `pagination-wrapper-center`
- `pagination-wrapper-right`

Alert box

Styles for `search:alert` tag.

- `alertbox`: Style element as an alert box
- `alertbox-secondary`: Secondary color
- `alertbox-positive`
- `alertbox-negative`
- `alertbox-warning`
- `alertbox-info`

Interruptors

Apply to any element or use the `block:interruptor` tag.

- `interruptor`: Style element as an attention grabber
  See the `message` example in the gallery.

Scroll to

Styles for the `widget:scroll-to` tag:

- `bottom-left`: Position bottom left of the page
- `bottom-right`
- `bottom-center`

Toggle panes

Styles to be used like:

```
<toggle:controls labels="Pane 1, Pane 2, Pane 3" panes="pills" styling="pill">
</toggle:controls> <toggle:panes id="pills"> <toggle:pane> ...
```

See *`toggle:controls`*.

- `tabs-shadow`: Adds shadow to tabs
- `collapsible`: Allows tabs to be collapsed by clicking the active tab
- `pill`: Pill style
- `line`
- `simple`
- `bordered`

Cards

- `basic-card`: Basic card style
- `card`: Default card style

Card results

The card widths are auto-generated. By default a maximum of 6 result cards can be shown on one row. By setting `@maxCardsPerRow` this can be increased or decreased.

On a *`search:result-list`* add the card styling in the following format:

```
cards-{size}-{numberPerRow}
```

For example:

```
<search:result-list styling="cards-sm-2 cards-lg-4">
```

This will show 2 cards per row on small screens, 4 on large screens.

Card styles

Styles for the *`search:result`* tag:

- `card-emphasis`
- `card-shadow`
- `card-step`

Popover

Styles for `widget:popover` tag:

- `popover-wrapper`: Popover style
- `popover-large`: Large popover
- `popover-medium`
- `popover-small`

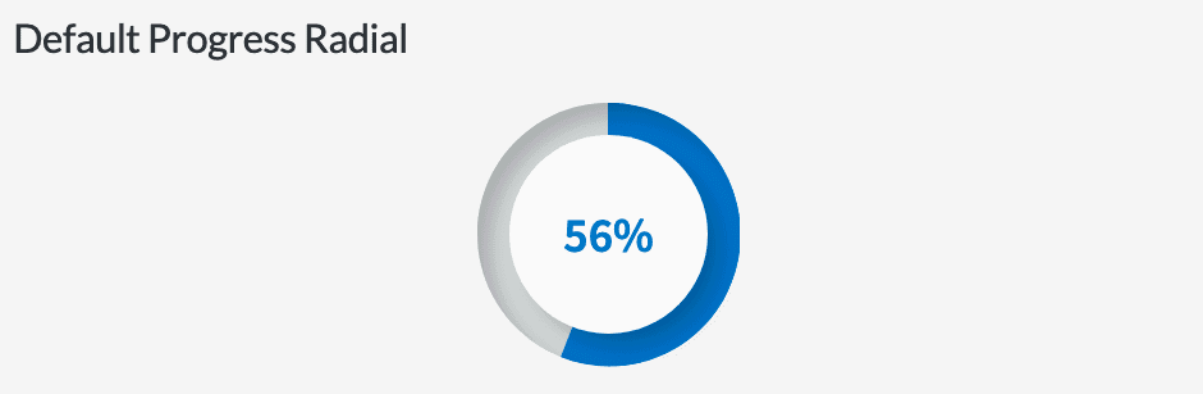
Tags

The App Studio markup language defines a set of HTML-like elements, or *tags*, for defining search queries to the backend and visualizing the search results and facets, handling all the complexities of connecting to Fusion.

For example, adding a radial progress bar is as simple including the following tag:

```
<progress:radial percentage="56"> </progress:radial>
```

This results in the following:



Over 150 tags are provided in the Appkit framework.

advanced:button

Description

Creates a button for triggering the advanced search.

Usage

as element:

```
<advanced:button [button-label="{string}"] [search-within-checkbox="{boolean}"] [search-within-label="{boolean}"] [search-within-default-value="{boolean}"] [action="{string}"]> </advanced:button>
```

Parameters

Param	Type	Details
button-label *(optional)*	**string**	Label for the button
search-within-checkbox *(optional)*	**boolean**	Show the 'search within' option as a checkbox. This enables the query to retain the filters already applied when checked. If not checked the search is reset when a query is entered and the form submitted. If you want this functionality but don't want a checkbox showing, you can use this attribute set to be false, and set `search-within-default-value` to be true - this uses a hidden field instead of a checkbox.
search-within-label *(optional)*	**boolean**	The label for the 'search within' checkbox.
search-within-default-value *(optional)*	**boolean**	Whether to enable 'search within' by default. The default is option is 'false'.
action *(optional)*	**string**	Page the search should take place on, results will not change automatically when an action is set.

advanced:date-range

Description

Creates an advanced search date range picker. Must be used within a `<search:advanced>` tag. A filter for the field specified is only set once both dates have been set.

Usage

as element:

```
<advanced:date-range field="{string}" values="{string}" [date-format="{string}"] [min-view="{string}"]> </advanced:date-range>
```

Parameters

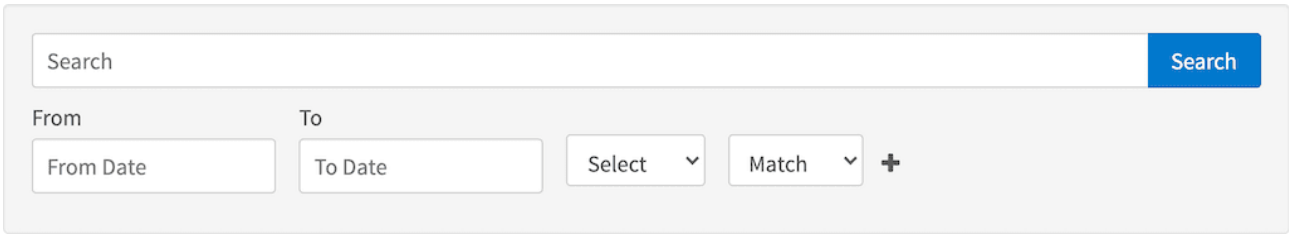
Param	Type	Details
field	**string**	The field to apply the filter to.
values	**string**	The checkbox values to display. These can be represented (comma separated) as field=displayName combinations and must match the actual value of a filter.
dateFormat *(optional)*	**string**	The display date format of the dates. Defaults to 'medium'.
minView *(optional)*	**string**	The minimum view of the date picker to display. Values: 'year/month/day/hour/minute'. Default 'minute'.

Example

Source

```
<search:advanced> <search:box></search:box> <advanced:date-range fields="source=Source,Material"></advanced:date-range> </search:advanced>
```

Result



advanced:text

Description

Creates an advanced search input field. Must be used within a <search:advanced> tag. The input value will be set as the filter value for the field specified.

Usage

as element:

```
<advanced:text field="{string}" label="{string}" placeholder="{string}" multiple>
</advanced:text>
```

Parameters

Param	Type	Details
field	**string**	The field to apply the filter to.
label	**string**	The input label.
placeholder	**string**	The input placeholder.
multiple	**boolean**	Allows additional input fields to be created by the user. Default true.

block:interruptor

Description

Creates an interrupter box with the transcluded content

Usage

as element:

```
<block:interruptor> </block:interruptor>
```

Example

Source

```
<block:interruptor> <h2>Heads Up</h2> <p>I'm interrupting you!</p>
</block:interruptor>
```

Result

EXAMPLE

> Heads Up
>
> I'm interrupting you!

```
<block:interruptor>
    <h2>Heads Up</h2>
    <p>I'm interrupting you!</p>
</block:interruptor>
```

chart:credit

Description

To display a credit on the chart in the bottom right hand corner use the Chart Credit tag with the text and href attributes. This tag must be nested in the *Chart Display tag*.

Usage

as element:

```
<chart:credit [text="{string}"] [href="{string}"]> </chart:credit>
```

Directive info

- This directive creates new scope.

Parameters

Param	Type	Details
text *(optional)*	**string**	The text for the credits label.
href *(optional)*	**string**	Url to redirect the user to when they click the credit.

Example

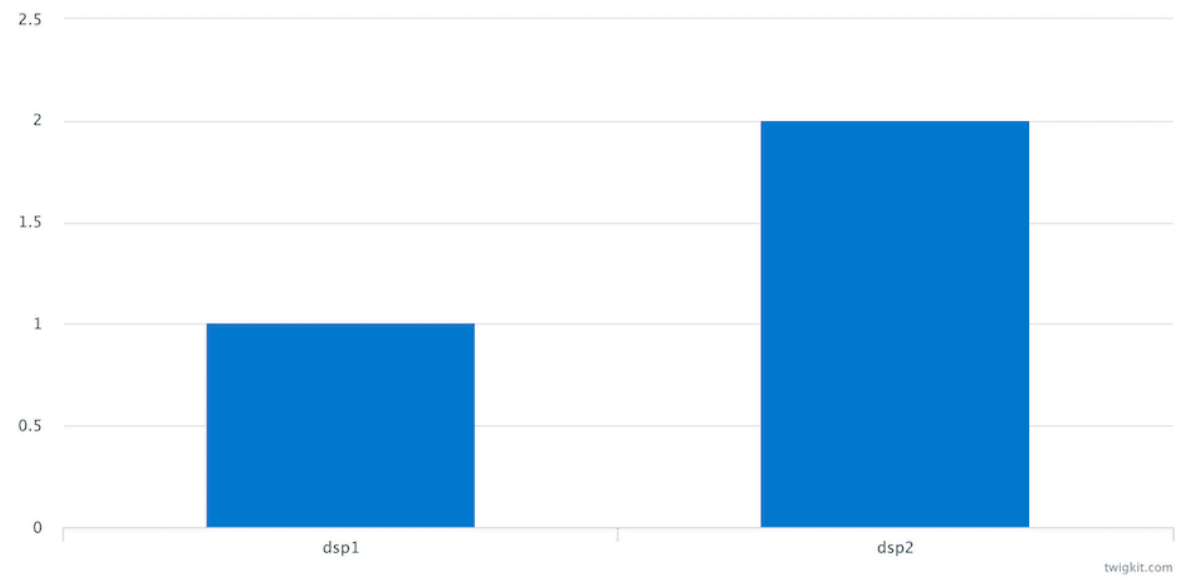
Source

```
<chart:display response="response" facet-name="test" type="column"> <chart:credit
text="twigkit.com" href="http://www.twigkit.com"></chart:credit> </chart:display>
```

```
angular.module('lightning') .controller('ExampleController',
['$scope','$timeout','ResponseService', function($scope,$timeout,ResponseService) {
$scope.response = { page: 2, query: { rpp: 50 }, facets: { test: { filters: [ {
val: { dsp: 'dsp1', act: 1 }, count: 1, }, { val: { dsp: 'dsp2', act: 2 }, count:
2, } ] }, }, results: [ { result: { fields: { label: 'label1', value: 1 } }, }, {
result: { fields: { label: 'label2', value: 2 } }, hits: { act: ['Hits', 264] } }]
} ResponseService.setResponse('response',$scope.response); }]);
```

Result

EXAMPLE




```
<chart:display response="response" facet-name="test" type="column">
   <chart:credit text="twigkit.com" href="http://www.twigkit.com"></chart:credit>
</chart:display>
```

chart:display

Description

The Chart Display Tag displays the facet provided in a chart. The body of the Chart Display tag can contain a *Series Facet Tag* for rendering multiple series using facets and also a *Series Result List Tag* for rendering multiple series using results. Along with this there are other tags to customise the chart, these include the *Chart Tooltip Tag* for modifying the tooltip, *Chart Legend Tag* for modifying the legend, *Chart Credit Tag* for adding a credit to the chart and the *Chart Export Tag* for exporting the chart.

For detail on how to change the color and background color see the demo below, for more detail on what you can change and set using CSS see here. *(http://www.highcharts.com/docs/chart-design-and-style/style-by-css)*

The following tags can be nested within the chart display tag.

- *Series Facet Tag*
- *Series Result List Tag*
- *Series Time Tag*
- *Chart Credit Tag*
- *Chart Export Tag*
- *Chart Legend Tag*
- *Chart Tooltip Tag*

Usage

as element:

```
<chart:display response="{string}" facet-name="{string}" type="{string}" [title="
{string}"] [subtitle="{string}"] [height="{string}"] [xtitle="{string}"] [ytitle="
{string}"] [stacking="{string}"] [radar-or-polar="{string}"] [xlabel-rotation="
{number}"] [max-characters="{number}"] [inner-size="{number}"] [invert-axes="
{boolean}"] [show-tooltips="{boolean}"] [query="{object}"] [options="{object}"]
[max-labels="{number}"] [show-labels="{boolean}"] [show-count="{boolean}"] [show-
data-labels="{boolean}"] [three-d="{boolean}"] [websocket="{string}"] [show-value="
{boolean}"] [select-series="{boolean}"] [select="{string}"] [single-filter-mode="
{boolean}"]> </chart:display>
```

Directive info

- This directive creates new scope.

Parameters

Param	Type	Details
response	**string**	The name of the response.
facet-name	**string**	Name of facet.
type	**string**	Chart type (use type of 'time' when using the series time tag).
title *(optional)*	**string**	Chart title
subtitle *(optional)*	**string**	Chart sub title
height *(optional)*	**string**	Chart height. Default:400 (400px but px is not required!)
xtitle *(optional)*	**string**	x axis title
ytitle *(optional)*	**string**	y axis title
stacking *(optional)*	**string**	Whether to stack the values of each series on top of each other when using multiple series. 'normal' to stack by value or 'percent'.
radar-or-polar *(optional)*	**string**	Set as 'radar' or 'polar' to turn your series into a radar/polar chart.
xlabel-rotation *(optional)*	**number**	Rotation of the labels in degrees

Param	Type	Details
max-characters *(optional)*	**number**	Number of characters to show in labels. default:250. (Cannot be used with a time series)
inner-size *(optional)*	**number**	The of size of the inner diameter of a pie chart. A size greater than 0 renders a donut chart.
invert-axes *(optional)*	**boolean**	Whether to invert the axes so that the x axis is vertical and y axis is horizontal. When true, the x axis is reversed by default. If a bar series is present in the chart, it will be inverted automatically. Defaults to false.
show-tooltips *(optional)*	**boolean**	Whether to display hover tooltips in the graph. Default:true
query *(optional)*	**object**	The search query from which to build links.
options *(optional)*	**object**	Extend the current options. Options can be found here. *(http://api.highcharts.com/highcharts/)*
max-labels *(optional)*	**number**	Maximum number of labels to show
show-labels *(optional)*	**boolean**	Whether to show labels in the graph, does not affect hover tooltips, default:true
show-count *(optional)*	**boolean**	Whether to show the count in the label. default:false

Param	Type	Details
show-data-labels *(optional)*	**boolean**	Whether to show the count on the point of the series (Not applicable to pie charts) default:false
three-d *(optional)*	**boolean**	Renders the chart in 3d. Only works with column and pie charts.
websocket *(optional)*	**string**	URL to receive updates from
show-value *(optional)*	**boolean**	Whether to show the value in the middle of the chart when hovered over. This will only work for pie charts with the `inner-size` attribute set greater then 0.
select-series *(optional)*	**boolean**	When set to `true` you will be able to select the series to show on the chart.

select
(optional)
	string	How the data should be filtered when clicking on a part of the chart, a valid query must be present. See *select example*
		`+`
		`Valid values:-`
		• **false**
		User selects a filter by clicking the value;
		(default) * **multiselect**
		Supports selecting multiple filters that get combined with an 'OR'. * **multiselect-instant**
		Supports selecting multiple filters that get combined with an 'OR', filters will be applied automatically.

single-filter-mode *(optional)*	**boolean**	When in single-filter mode, a facet visualisation will be shown as a single number, in large print, when the facet being rendered has only a single filter value. This is to avoid showing a single bar, column or pie slice when rendering a visualisation over a single data bucket. See *single filter mode example*

Example

Source

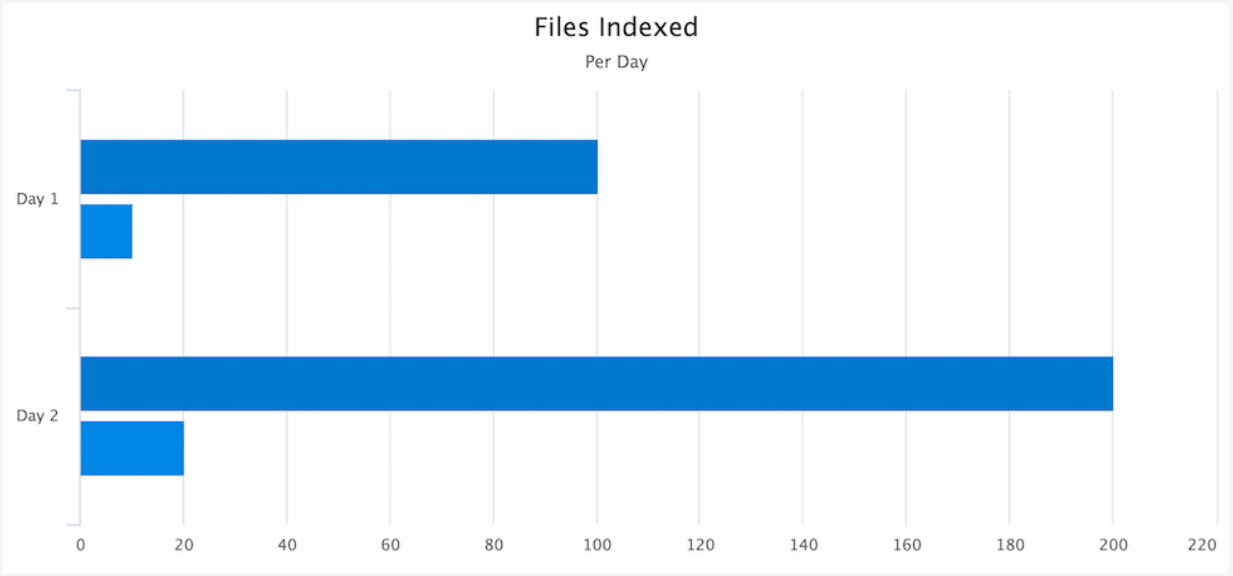
```
<div ng-controller="ExampleController"> <div> <h3 id="example_chart-types">Chart
Types</h3> <toggle:controls labels="Bar, Column, Pie, Donut, Donut 2, Line, Area,
Area Spline" panes="panes"></toggle:controls> <toggle:panes id="panes">
<toggle:pane> <chart:display response="response" facet-name="population" type="bar"
title="Population of Countries" subtitle="Bar Chart"> <chart:credit
text="twigkit.com" href="http://www.twigkit.com"></chart:credit> <chart:export
file-name="bar"></chart:export> </chart:display> </toggle:pane> <toggle:pane>
<chart:display response="response" facet-name="population" type="column"
title="Population of Countries" subtitle="Column Chart"> <chart:credit
text="twigkit.com" href="http://www.twigkit.com"></chart:credit> <chart:export
file-name="column"></chart:export> </chart:display> </toggle:pane> <toggle:pane>
<chart:display response="response" facet-name="population" type="pie"
title="Population of Countries" subtitle="Pie Chart"> <chart:credit
text="twigkit.com" href="http://www.twigkit.com"></chart:credit> <chart:export
file-name="pie"></chart:export> </chart:display> </toggle:pane> <toggle:pane>
<chart:display response="response" facet-name="population" type="pie" inner-
size="80%" show-tooltips="false" show-value="true" title="Population of Countries"
subtitle="Donut Chart"> <chart:credit text="twigkit.com"
href="http://www.twigkit.com"></chart:credit> <chart:export file-name="donut">
</chart:export> </chart:display> </toggle:pane> <toggle:pane> <chart:display
response="response" facet-name="population" type="donut" title="Population of
Countries" subtitle="Donut Chart 2"> <chart:credit text="twigkit.com"
href="http://www.twigkit.com"></chart:credit> <chart:export file-name="donut">
</chart:export> </chart:display> </toggle:pane> <toggle:pane> <chart:display
response="response" facet-name="population" type="line" title="Population of
Countries" subtitle="Line Chart"> <chart:credit text="twigkit.com"
href="http://www.twigkit.com"></chart:credit> <chart:export file-name="line">
</chart:export> </chart:display> </toggle:pane> <toggle:pane> <chart:display
response="response" facet-name="population" type="area" title="Population of
Countries" subtitle="Area Chart"> <chart:credit text="twigkit.com"
href="http://www.twigkit.com"></chart:credit> <chart:export file-name="area">
</chart:export> </chart:display> </toggle:pane> <toggle:pane> <chart:display
response="response" facet-name="population" type="areaspline" title="Population of
Countries" subtitle="Area Spline Chart"> <chart:credit text="twigkit.com"
href="http://www.twigkit.com"></chart:credit> <chart:export file-name="areaspline">
</chart:export> </chart:display> </toggle:pane> </toggle:panes> </div> <hr/> <div>
<h3 id="example_radar-chart-type">Radar Chart Type</h3> <toggle:controls
labels="Bar, Line, Area" panes="radar"></toggle:controls> <toggle:panes id="radar">
<toggle:pane> <chart:display response="response" facet-name="population" type="bar"
title="Population of Countries" subtitle="Bar Chart" radar-or-polar="radar">
<chart:credit text="twigkit.com" href="http://www.twigkit.com"></chart:credit>
<chart:export file-name="bar"></chart:export> </chart:display> </toggle:pane>
<toggle:pane> <chart:display response="response" facet-name="population"
type="line" title="Population of Countries" subtitle="Line Chart" radar-or-
polar="radar"> <chart:credit text="twigkit.com" href="http://www.twigkit.com">
</chart:credit> <chart:export file-name="line"></chart:export> </chart:display>
</toggle:pane> <toggle:pane> <chart:display response="response" facet-
name="population" type="area" title="Population of Countries" subtitle="Area Chart"
radar-or-polar="radar"> <chart:credit text="twigkit.com"
href="http://www.twigkit.com"></chart:credit> <chart:export file-name="area">
</chart:export> </chart:display> </toggle:pane> </toggle:panes> </div> <hr/> <div>
<h3 id="example_polar-chart-type">Polar Chart Type</h3> <toggle:controls
labels="Bar, Line, Area" panes="polar"></toggle:controls> <toggle:panes id="polar">
<toggle:pane> <chart:display response="response" facet-name="population" type="bar"
title="Population of Countries" subtitle="Bar Chart" radar-or-polar="polar">
<chart:credit text="twigkit.com" href="http://www.twigkit.com"></chart:credit>
<chart:export file-name="bar"></chart:export> </chart:display> </toggle:pane>
<toggle:pane> <chart:display response="response" facet-name="population"
type="line" title="Population of Countries" subtitle="Line Chart" radar-or-
polar="polar"> <chart:credit text="twigkit.com" href="http://www.twigkit.com">
</chart:credit> <chart:export file-name="line"></chart:export> </chart:display>
```

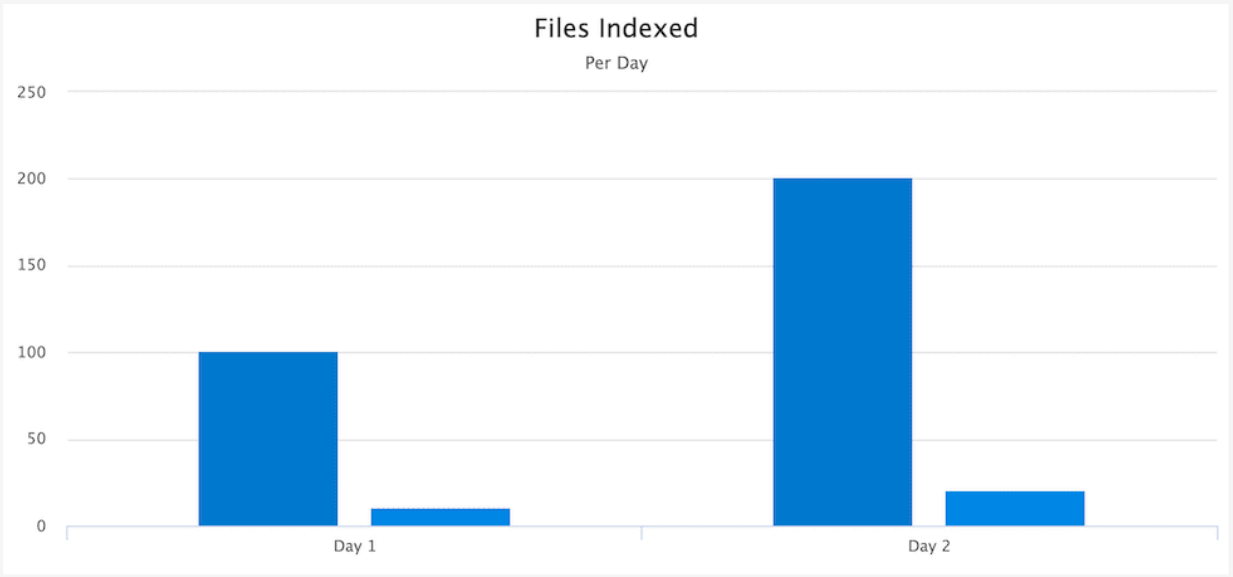
```
</toggle:pane> <toggle:pane> <chart:display response="response" facet-
name="population" type="area" title="Population of Countries" subtitle="Area Chart"
radar-or-polar="polar"> <chart:credit text="twigkit.com"
href="http://www.twigkit.com"></chart:credit> <chart:export file-name="area">
</chart:export> </chart:display> </toggle:pane> </toggle:panes> </div> <hr/> <div>
<h3 id="example_axis-titles">Axis Titles</h3> <chart:display response="response"
facet-name="population" type="bar" title="Population of Countries" xtitle="Country"
ytitle="Population"> <chart:credit text="twigkit.com"
href="http://www.twigkit.com"></chart:credit> <chart:export file-name="column">
</chart:export> </chart:display> </div> <hr/> <div> <h3 id="example_3d">3D</h3>
<chart:display response="response" facet-name="population" type="pie"
title="Population of Countries" three-d="true"> <chart:credit text="twigkit.com"
href="http://www.twigkit.com"></chart:credit> </chart:display> <hr/> <chart:display
response="response" facet-name="population" type="column" title="Population of
Countries" three-d="true"> <chart:credit text="twigkit.com"
href="http://www.twigkit.com"></chart:credit> </chart:display> </div> <hr/> <div>
<h3 id="example_label-rotation">Label Rotation</h3> <chart:display
response="response" facet-name="population" type="column" title="Population of
Countries" xlabel-rotation="45"> <chart:credit text="twigkit.com"
href="http://www.twigkit.com"></chart:credit> </chart:display> <hr/> <chart:display
response="response" facet-name="population" type="column" title="Population of
Countries" xlabel-rotation="-45"> <chart:credit text="twigkit.com"
href="http://www.twigkit.com"></chart:credit> </chart:display> </div> <hr/> <div>
<h3 id="example_show-labels">Show Labels (false)</h3> <chart:display
response="response" facet-name="population" type="pie" title="Population of
Countries" show-labels="false"> <chart:credit text="twigkit.com"
href="http://www.twigkit.com"></chart:credit> </chart:display> <hr/> <chart:display
response="response" facet-name="population" type="pie" inner-size="80%" show-
value="true" show-tooltips="false" title="Population of Countries" show-
labels="false"> <chart:credit text="twigkit.com" href="http://www.twigkit.com">
</chart:credit> </chart:display> </div> <div> <h3 id="example_custom">Custom (<a
href="http://api.highcharts.com/highcharts/" target="_blank">Options</a>)</h3>
<chart:display response="response" type="line" title="Population of Countries"
xlabel-rotation="-45" options="custom"> <series:facet response="response"
name="population" title="Population Present Day"></series:facet> <series:facet
response="response" name="population2000" title="Population 2000"></series:facet>
<chart:credit text="twigkit.com" href="http://www.twigkit.com"></chart:credit>
</chart:display> <h4 id="example_custom_zoom">Zoom (Custom)</h4> <p>Custom
Object</p> <pre>{{zoomCustom | json}}</pre> <p>Use your mouse to select the area
you wish to zoom in on</p> <chart:display response="response" type="line"
title="Population of Countries" options="zoomCustom"> <series:facet
response="response" name="population" title="Population Present Day">
</series:facet> <series:facet response="response" name="population2000"
title="Population 2000"></series:facet> <chart:credit text="twigkit.com"
href="http://www.twigkit.com"></chart:credit> </chart:display> </div> <div
id="colour-demo"> <h3 id="example_colour">Colour (since 0.3.0)</h3> <p>Add an ID to
the element wrapping your chart and configure the background colour and series
colours like so.</p> <pre> #colour-demo .highcharts-color-0{ fill:pink;
stroke:pink; } #colour-demo .highcharts-color-1{ fill:yellow; stroke:yellow; }
#colour-demo .highcharts-background{ fill:#f5f0f0; } </pre> <p>For more detail on
what you can change and set using CSS see <a
href="http://www.highcharts.com/docs/chart-design-and-style/style-by-css"
target="_blank">here.</a></p> <chart:display response="response" type="line"
title="Population of Countries" xlabel-rotation="-45" options="custom">
<series:facet response="response" name="population" title="Population Present Day">
</series:facet> <series:facet response="response" name="population2000"
title="Population 2000"></series:facet> <chart:credit text="twigkit.com"
href="http://www.twigkit.com"></chart:credit> </chart:display> </div> <div> <h3
id="example_selectable-series">Selectable Series</h3> <chart:display
response="response" type="line" title="Population of Countries" xlabel-
rotation="-45" select="checkbox"> <series:facet response="response"
name="population" title="Population Present Day" selected-by-default="false">
</series:facet> <series:facet response="response" name="population2000"
title="Population 2000"></series:facet> <chart:credit text="twigkit.com"
href="http://www.twigkit.com"></chart:credit> </chart:display> </div> </div>
```

```
#colour-demo .highcharts-color-0{ fill:pink; stroke:pink; colour-demo .highcharts-color-1{ fill:yellow; stroke:yellow; colour-demo .highcharts-background{ fill:#f5f0f0;
```

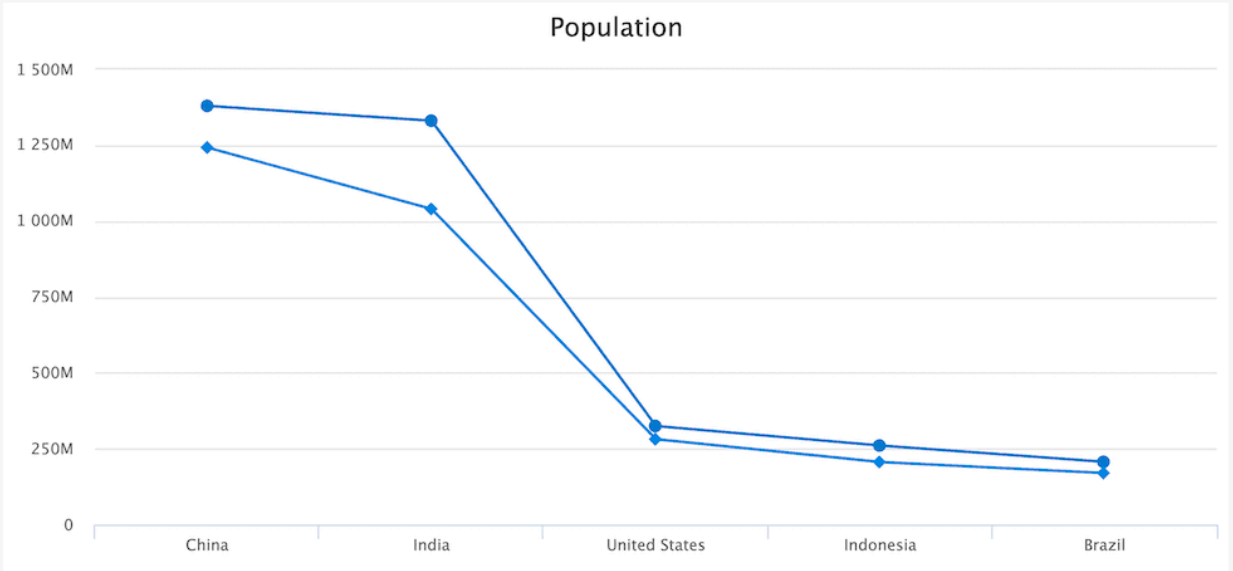
```
angular.module('lightning') .controller('ExampleController', ['$scope','$timeout', 'ResponseService', function($scope,$timeout, ResponseService) { $scope.custom = { tooltip:{ shared:true, split:true }, xAxis: { plotLines: [{ value: 3, // Value of where the line will appear width: 1 // Width of the line }] } }; $scope.zoomCustom = { chart:{ zoomType:'xy' } } $scope.response = { page: 2, query: { rpp: 50 }, facets: { test: { filters: [ { val: { dsp: 'dsp1', act: 1 }, count: 1, }, { val: { dsp: 'dsp2', act: 2 }, count: 2, } ] }, population:{ filters: [ { val: 'China', count: 1379790000, }, { val: 'India', count: 1330780000, }, { val: 'United States', count: 324897000, }, { val: 'Indonesia', count: 260581000, }, { val: 'Brazil', count: 206927000, }, { val: 'Pakistan', count: 194772000, }, { val: 'Nigeria', count: 186987000, }, { val: 'Bangladesh', count: 161401000, }, { val: 'Russia', count: 146727405, }, { val: 'Japan', count: 126693000, }, ] }, population2000:{ filters: [ { val: 'China', count: 1242612300, }, { val: 'India', count: 1040000000, }, { val: 'United States', count: 281421923, }, { val: 'Indonesia', count: 206264595, }, { val: 'Brazil', count: 170000000, }, { val: 'Pakistan', count: 140000000, }, { val: 'Nigeria', count: 110000000, }, { val: 'Bangladesh', count: 130000000, }, { val: 'Russia', count: 147000000, }, { val: 'Japan', count: 127000000, }, ] }, donut: { filters: [ { val: { dsp: 'Short', act: 1 }, count: 1, }, { val: { dsp: 'Long Donut Name(15)', act: 2 }, count: 2, }, { val: { dsp: 'Long Long Donut Name', act: 10 }, count: 10, }, { val: { dsp: 'Long Long Long Donut Name', act: 20 }, count: 20, }, { val: { dsp: 'Long Long Long Long Donut Name', act: 20 }, count: 20, }, { val: { dsp: 'Long Long Long Long Long Donut Name', act: 20 }, count: 20, }, { val: { dsp: 'Long Long Long Long Long Long Donut Name', act: 20 }, count: 20, } ] }, }, results: [ { result: { fields: { label:{ val:'label1', } , value:{ val:1, }, anotherValue:{ val:1, }, } }, }, { result: { fields: { label:{ val:'label2', } , value:{ val:2, }, anotherValue:{ val:100, }, } }, hits: { act: ['Hits', 264] } }] } ResponseService.setResponse('response',$scope.response); }]);
```

Result

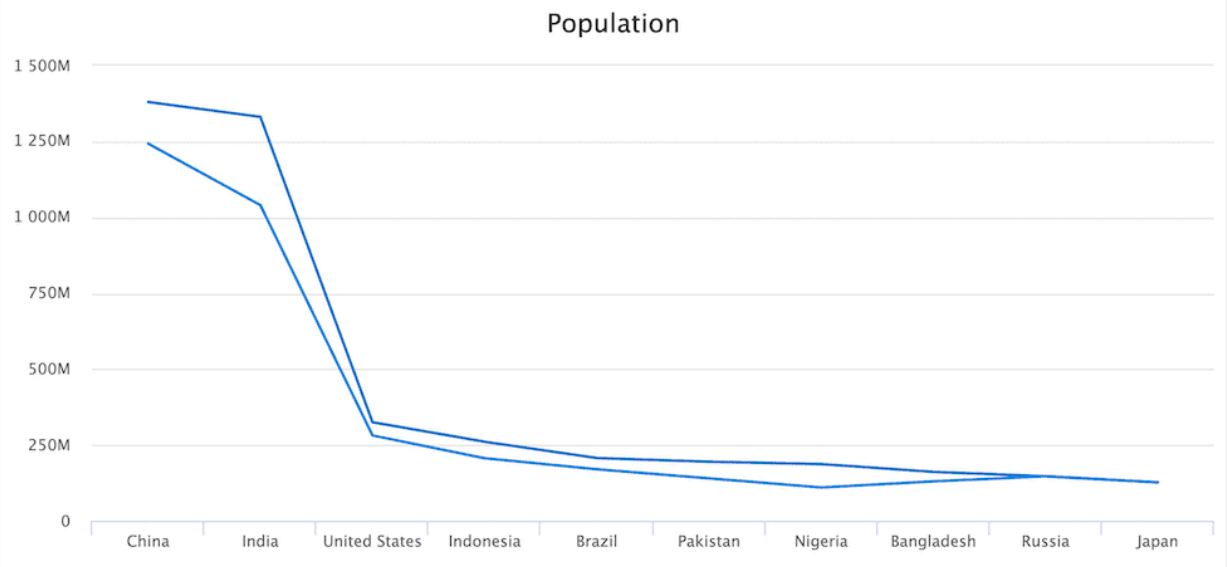




Number To Show Attribute



Show Marker Attribute



Colour (since 0.3.0)

Add an ID to the element wrapping your chart and configure the background colour and series colours like so.

```
#colour-demo .highcharts-color-0{
  fill:pink;
  stroke:pink;
}
#colour-demo .highcharts-color-1{
  fill:yellow;
  stroke:yellow;
}
#colour-demo .highcharts-background{
  fill:#f5f0f0;
}
```

For more detail on what you can change and set using CSS see here.

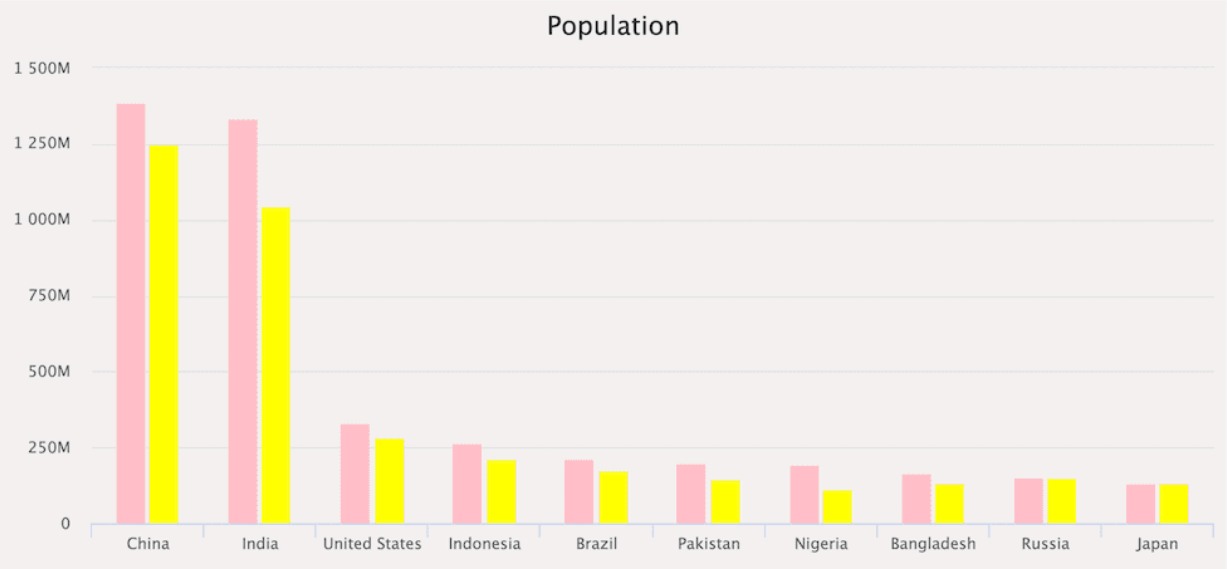


chart:export

Description

To give users the option to export a chart include the Chart Export tag within the Chart Display tag. Adding the tag will enable exporting without adding any attributes. This tag must be nested in the *Chart Display tag*.

Usage

as element:

```
<chart:export [file-name="{string}"] [url="{string}"]> </chart:export>
```

Directive info

- This directive creates new scope.

Parameters

Param	Type	Details
file-name *(optional)*	**string**	Custom name for the file when downloaded, if nothing is entered it will default to 'chart'.
url *(optional)*	**string**	Url of the export server, if left blank it will use the providers export server.

Example

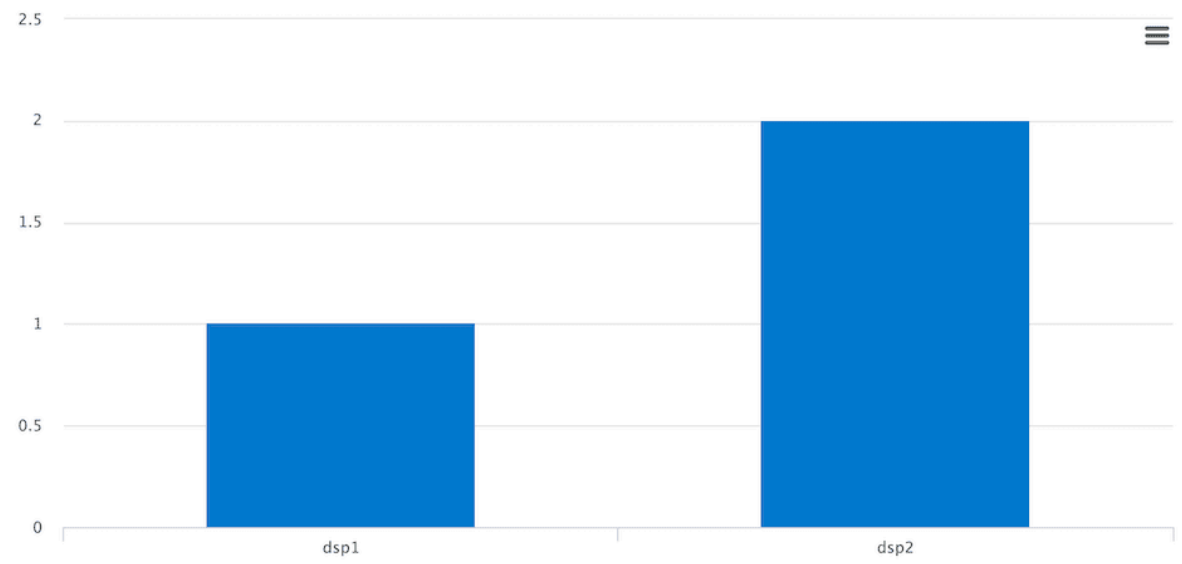
Source

```
<chart:display response="response" facet-name="test" type="column"> <chart:export
file-name="column"></chart:export> </chart:display>
```

```
angular.module('lightning') .controller('ExampleController',
['$scope','$timeout','ResponseService', function($scope,$timeout,ResponseService) {
$scope.response = { page: 2, query: { rpp: 50 }, facets: { test: { filters: [ {
val: { dsp: 'dsp1', act: 1 }, count: 1, }, { val: { dsp: 'dsp2', act: 2 }, count:
2, } ] }, }, results: [ { result: { fields: { label: 'label1', value: 1 } }, }, {
result: { fields: { label: 'label2', value: 2 } }, hits: { act: ['Hits', 264] } }]
} ResponseService.setResponse('response',$scope.response); }]);
```

Result

EXAMPLE




```
<chart:display response="response" facet-name="test" type="column">
    <chart:export file-name="column"></chart:export>
</chart:display>
```

chart:gauge

Description

The chart gauge directive generates a gauge, you can also use the chart export and chart credit directives to add this functionality to your gauge.

Usage

as element:

```
<chart:gauge value="{number}" min="{number}" max="{number}" [stops="{array}"]
[title="{string}"] [value-suffix="{string}"] [height="{string}"] [options="
{object}"]> </chart:gauge>
```

Directive info

- This directive creates new scope.

Parameters

Param	Type	Details
value	**number**	Value to be displayed on the gauge.
min	**number**	The smallest number that can be displayed on the gauge.
max	**number**	The highest number that can be displayed on the gauge.
stops *(optional)*	**array**	The colours to render at each stop measured from 0 to 1. Default [[0.1, '#DF5353'], // red [0.5, '#DDDF0D'], // yellow [0.9, '#55BF3B'] // green]
title *(optional)*	**string**	The title of the chart.
value-suffix *(optional)*	**string**	Suffix for the value on the gauge.
height *(optional)*	**string**	Chart height. Default:400 (400px but px is not required!)
options *(optional)*	**object**	Extend the current options. Options can be found here. *(http://api.highcharts.com/highcharts/)*

Example

Source

```
<div ng-controller="ExampleController"> <chart:gauge value="{{value}}" min="
{{min}}" max="{{max}}" title="Speed" value-suffix="mph" stops="[[0.1, '#55BF3B'],
[0.5, '#DDDF0D'], [0.9, '#DF5353']]"> <chart:credit text="twigkit.com"
href="http://www.twigkit.com"></chart:credit> <chart:export file-name="gauge">
</chart:export> </chart:gauge> <form class="tk-stl-form tk-stl-form-aligned">
<fieldset> <div class="tk-stl-control-group"> <label for="value">Value</label>
<input id="value" type="number" placeholder="Value" ng-model="value"> <button ng-
click="generateNewNumber()" style="button tiny">Generate Random Value</button>
</div> <div class="tk-stl-control-group"> <label for="min">Min</label> <input
id="min" type="number" placeholder="Min" ng-model="min"> </div> <div class="tk-stl-
control-group"> <label for="max">Max</label> <input id="max" type="number"
placeholder="Max" ng-model="max"> </div> </fieldset> </form> </div>
```

```
angular.module('lightning') .controller('ExampleController', ['$scope','$timeout',
function($scope,$timeout) { $scope.value = 20; $scope.min = 0; $scope.max = 40
$scope.generateNewNumber = function(){ $scope.value = Math.floor(Math.random() *
($scope.max - $scope.min + 1)) + $scope.min; }; }]);
```

Result

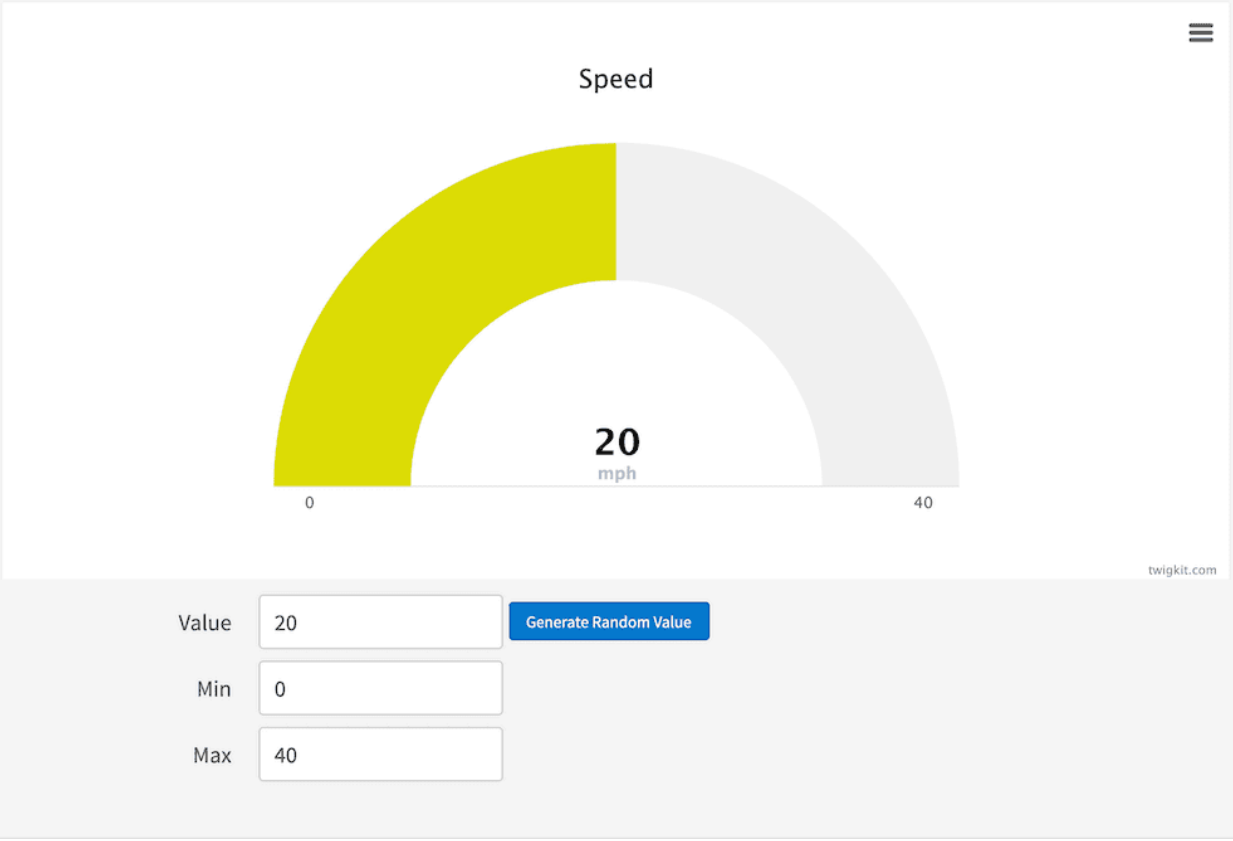


chart:histogram

Description

The Chart Histogram Directive

Usage

as element:

```
<chart:histogram [response="{object}"] [facet-name="{string}"] [query="{string}"]
[title="{string}"] [show="{number}"] [height="{string}"] [show-count="{boolean}"]
[max-characters="{number}"] [action="{string}"] [date-format="{string}"]>
</chart:histogram>
```

Directive info

- This directive creates new scope.

Parameters

Param	Type	Details
response *(optional)*	**object**	A response object
facet-name *(optional)*	**string**	Facet name to use from the response object
query *(optional)*	**string**	The name of the query to build links from.
title *(optional)*	**string**	The title that should be displayed above the histogram.
show *(optional)*	**number**	The maximum number of bars to show in the histogram. Defaults to 20.
height *(optional)*	**string**	Height of the chart specified by a valid CSS height style value (e.g. '100px' or '7em'). Defaults to '100px'
show-count *(optional)*	**boolean**	Whether to display the count in the label for each bar
max-characters *(optional)*	**number**	The number of characters to show before truncating the label. Default: 20.
action *(optional)*	**string**	The target resource to which the histogram query should submit.
date-format *(optional)*	**string**	How to format the date. See #/api/lightning.filter:dateFormat[date format filter] for valid formats.

Example

Source

```
<div ng-controller="ExampleController"> <h3 id="example_histogram">Histogram</h3>
<chart:histogram response="response" facet-name="test1"></chart:histogram> <hr />
<h3 id="example_histogram-with-date-formatting">Histogram with Date Formatting</h3>
<chart:histogram response="response" facet-name="test3" date-format="fullDate">
</chart:histogram> <hr /> <h3 id="example_large-dateset">Large Dateset</h3>
<chart:histogram response="response" facet-name="date" date-format="fullDate"
show="99" show-count="true"></chart:histogram> <widget:slider response="response"
facet-name="date" date-format="fullDate" sort="false"></widget:slider> </div>
```

```
angular.module('lightning') .controller('ExampleController',
['$scope','$timeout','ResponseService', function($scope,$timeout,ResponseService) {
$scope.response = { page: 2, query: { rpp: 50, filters:[] }, facets: { test1: {
filters: [ { val: { dsp: 'Day 1', act: 'Day 1', }, count: 100, }, { val: { dsp:
'Day 2', act: 'Day 2', }, count: 200, } ], 'max-count':'200' }, test2: { filters: [
{ val: { dsp: 'Day 1', act: 'Day 1', }, count: 10, }, { val: { dsp: 'Day 2', act:
'Day 2', }, count: 20, } ], 'max-count':20 }, test3: { filters: [ { val: { dsp:
1288323623006, act: 1288323623006, }, count: 10, }, { val: { dsp: 1288323723006,
act: 1288323723006, }, count: 20, } ], 'max-count':20 }, date:{ filters:[], 'max-
count':100 } }, results: [] } var i = 0; while(i < 99){ var CurrentDate = new
Date(); CurrentDate.setMonth(CurrentDate.getMonth() + i);
$scope.response.facets.date.filters.push( { val : { dsp:CurrentDate,
act:CurrentDate }, count:Math.floor(Math.random()*(100-1+1)+1), } ) i++ }
console.log($scope.response.facets.date); ResponseService.setResponse('response',
$scope.response); }]);
```

Result

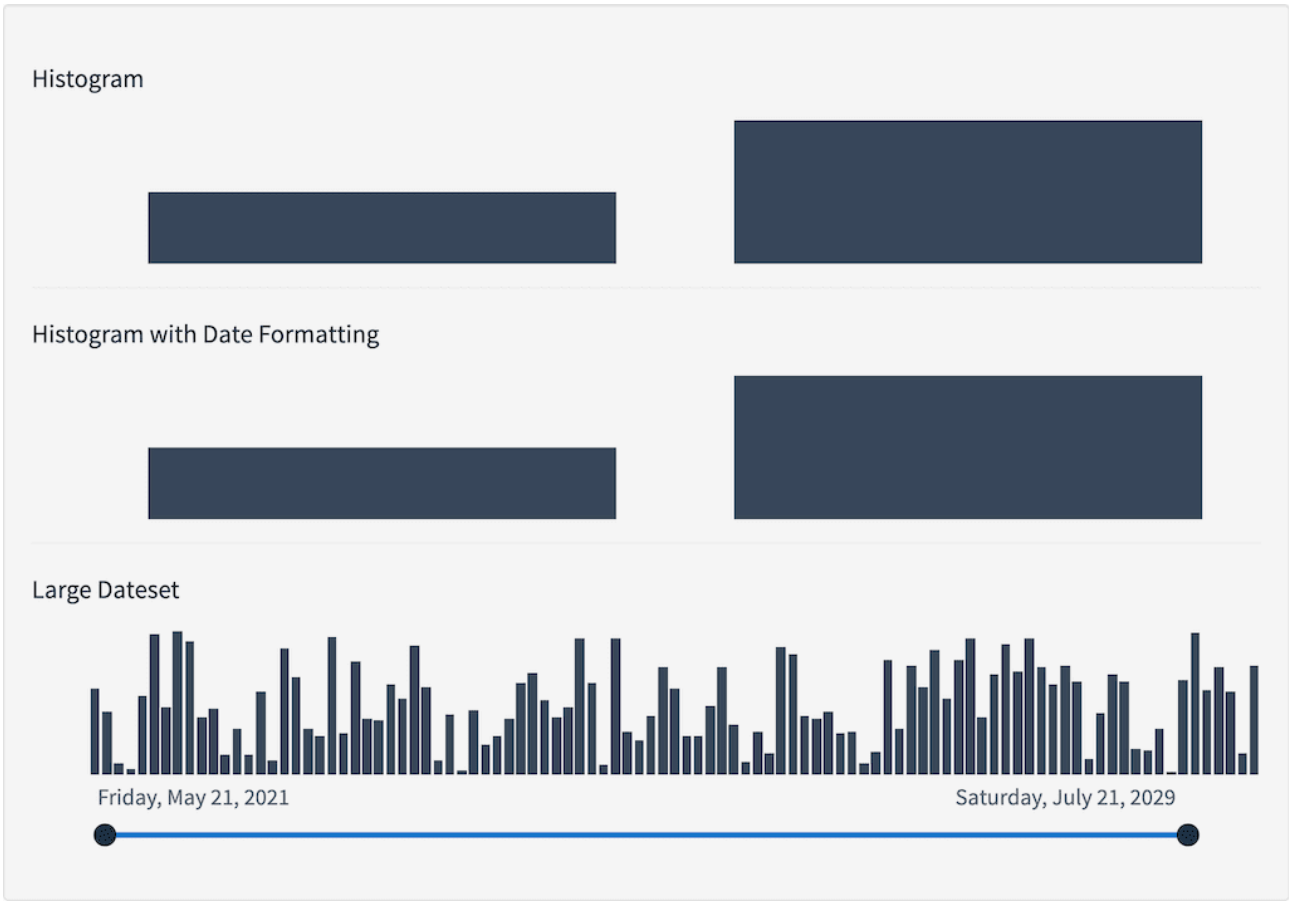

Histogram
Histogram with Date Formatting
Large Dateset
Friday, May 21, 2021
Saturday, July 21, 2029

chart:legend

Description

A legend is displayed when the chart has multiple series, with the Chart Legend tag you can customise the legend with a title, border and location on the chart along with other things. This tag must be nested in the *Chart Display tag*.

Usage

as element:

```
<chart:legend [title="{string}"] [align="{string}"] [vertical-align="{string}"] [layout="{string}"] [x="{string}"] [y="{string}"]> </chart:legend>
```

Directive info

- This directive creates new scope.

Parameters

Param	Type	Details
title *(optional)*	**string**	The title of the legend.
align *(optional)*	**string**	The horizontal alignment of the legend box within the chart area. Valid values are 'left', 'center' and 'right'.
vertical-align *(optional)*	**string**	The vertical alignment of the legend box. Valid values are 'top', 'middle' or 'bottom'.
layout *(optional)*	**string**	The layout of the legend items. Can be one of 'horizontal' or 'vertical'.
x *(optional)*	**string**	The x offset of the legend relative to its horizontal alignment 'align'. Negative x moves it to the left, positive x moves it to the right.
y *(optional)*	**string**	The vertical offset of the legend relative to it’s vertical alignment 'verticalAlign'. Negative y moves it up, positive y moves it down.

Example

Source

```
<div ng-controller="ExampleController"> <h3 id="example_legend">Legend</h3> <h4
id="example_legend_legend-on-the-right">Legend on the Right</h4> <tk-example>
<chart:display response="response" title="Files Indexed" subtitle="Per Day">
<series:facet name="test1" type="column" title="First Pass"></series:facet>
<series:facet name="test2" type="column" title="Second Pass"></series:facet>
<chart:legend title="Legend Title" align="right" vertical-align="top"
layout="vertical" y="100"></chart:legend> </chart:display> </tk-example> <tk-code-
wrapper> <code hljs> <chart:display response="response" title="Files Indexed"
subtitle="Per Day"> <series:facet name="test1" type="column" title="First Pass">
</series:facet> <series:facet name="test2" type="column" title="Second Pass">
</series:facet> <chart:legend title="Legend Title" align="right" vertical-
align="top" layout="vertical" y="100"></chart:legend> </chart:display> </code>
</tk-code-wrapper> <hr/> <h4 id="example_legend_legend-on-the-left">Legend on the
Left</h4> <tk-example> <chart:display response="response" title="Files Indexed"
subtitle="Per Day"> <series:facet name="test1" type="column" title="First Pass">
</series:facet> <series:facet name="test2" type="column" title="Second Pass">
</series:facet> <chart:legend title="Legend Title" align="left" vertical-
align="top" layout="vertical" y="100"></chart:legend> <chart:legend title="Legend
Title" align="left" vertical-align="top" layout="vertical" y="100"></chart:legend>
</chart:display> </tk-example> <tk-code-wrapper> <code hljs> <chart:display
response="response" title="Files Indexed" subtitle="Per Day"> <series:facet
name="test1" type="column" title="First Pass"></series:facet> <series:facet
name="test2" type="column" title="Second Pass"></series:facet> <chart:legend
title="Legend Title" align="right" vertical-align="top" layout="vertical" y="100">
</chart:legend> </chart:display> </code> </tk-code-wrapper> <hr/> <h4
id="example_legend_legend-on-the-bottom">Legend on the Bottom</h4> <tk-example>
<chart:display response="response" title="Files Indexed" subtitle="Per Day">
<series:facet name="test1" type="column" title="First Pass"></series:facet>
<series:facet name="test2" type="column" title="Second Pass"></series:facet>
<chart:legend title="Legend Title" align="center" vertical-align="bottom"
layout="horizontal"</chart:legend> </chart:display> </tk-example> <tk-code-wrapper>
<code hljs> <chart:display response="response" title="Files Indexed" subtitle="Per
Day"> <series:facet name="test1" type="column" title="First Pass"></series:facet>
<series:facet name="test2" type="column" title="Second Pass"></series:facet>
<chart:legend title="Legend Title" align="center" vertical-align="bottom"
layout="vertical"></chart:legend> </chart:display> </code> </tk-code-wrapper> <hr/>
<h4 id="example_legend_adding-a-border">Adding a Border</h4> <p>Add an ID or class
to the element wrapping your chart and configure the stroke width to add a border
and the value rx to modify the border radius.</p> <pre> .chart-with-legend-border
.highcharts-legend-box { fill: none; stroke-width: 1px; stroke: black; rx:2; }
</pre> <div class="chart-with-legend-border"> <tk-example> <chart:display
response="response" title="Files Indexed" subtitle="Per Day"> <series:facet
name="test1" type="column" title="First Pass"></series:facet> <series:facet
name="test2" type="column" title="Second Pass"></series:facet> <chart:legend
title="Legend Title" align="right" vertical-align="middle" layout="vertical"">
</chart:legend> </chart:display> </tk-example> <tk-code-wrapper> <code hljs>
<chart:display response="response" title="Files Indexed" subtitle="Per Day">
<series:facet name="test1" type="column" title="First Pass"></series:facet>
<series:facet name="test2" type="column" title="Second Pass"></series:facet>
<chart:legend title="Legend Title" align="right" vertical-align="middle"
layout="vertical"></chart:legend> </chart:display> </code> </tk-code-wrapper>
</div> </div>
```

```
.chart-with-legend-border .highcharts-legend-box { fill: none; stroke-width: 1px; stroke: black; rx:2; }
```

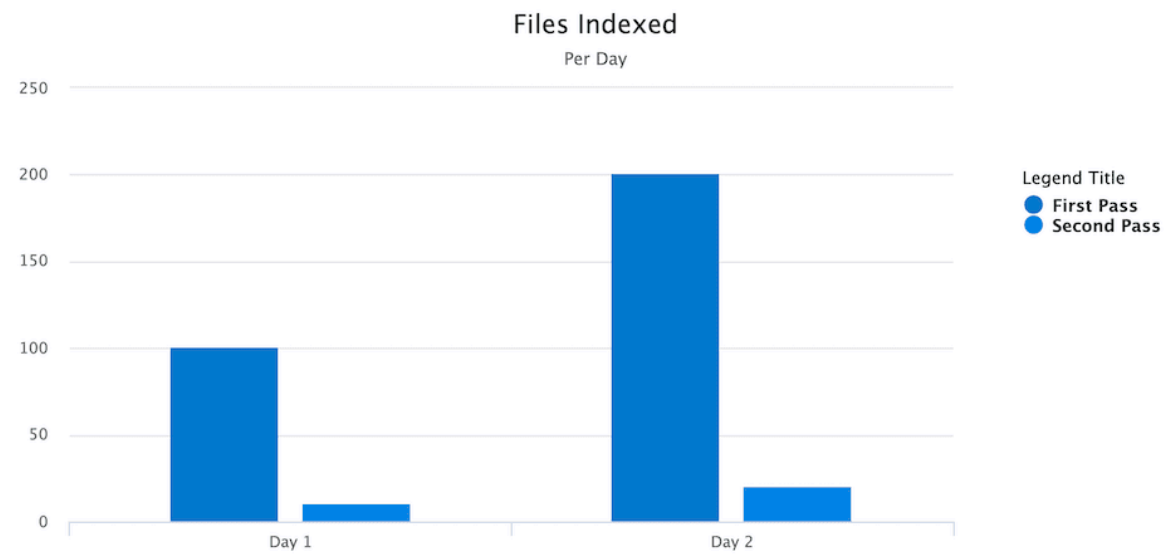
```
angular.module('lightning') .controller('ExampleController', ['$scope','$timeout','ResponseService', function($scope,$timeout, ResponseService) { $scope.response = { page: 2, query: { rpp: 50 }, facets: { test1: { filters: [ { val: { dsp: 'Day 1', act: 'Day 1', }, count: 100, }, { val: { dsp: 'Day 2', act: 'Day 2', }, count: 200, } ] }, test2: { filters: [ { val: { dsp: 'Day 1', act: 'Day 1', }, count: 10, }, { val: { dsp: 'Day 2', act: 'Day 2', }, count: 20, } ] }, }, results: [ { result: { fields: { label: 'label1', value: 1 } }, }, { result: { fields: { label: 'label2', value: 2 } }, hits: { act: ['Hits', 264] } }] } ResponseService.setResponse('response',$scope.response); }]);
```

Result

Legend

Legend on the Right

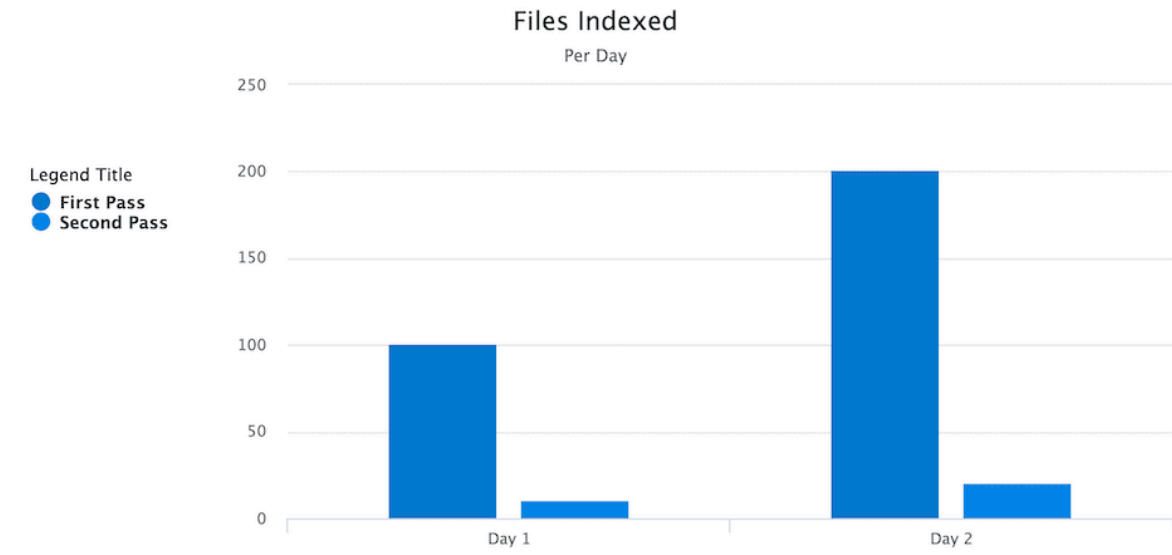
EXAMPLE




```
<chart:display response="response" title="Files Indexed" subtitle="Per Day">
    <series:facet name="test1" type="column" title="First Pass"></series:facet>
    <series:facet name="test2" type="column" title="Second Pass"></series:facet>
    <chart:legend title="Legend Title" align="right" vertical-align="top" layout="vertical"
y="100"></chart:legend>
</chart:display>
```

Legend on the Left

EXAMPLE

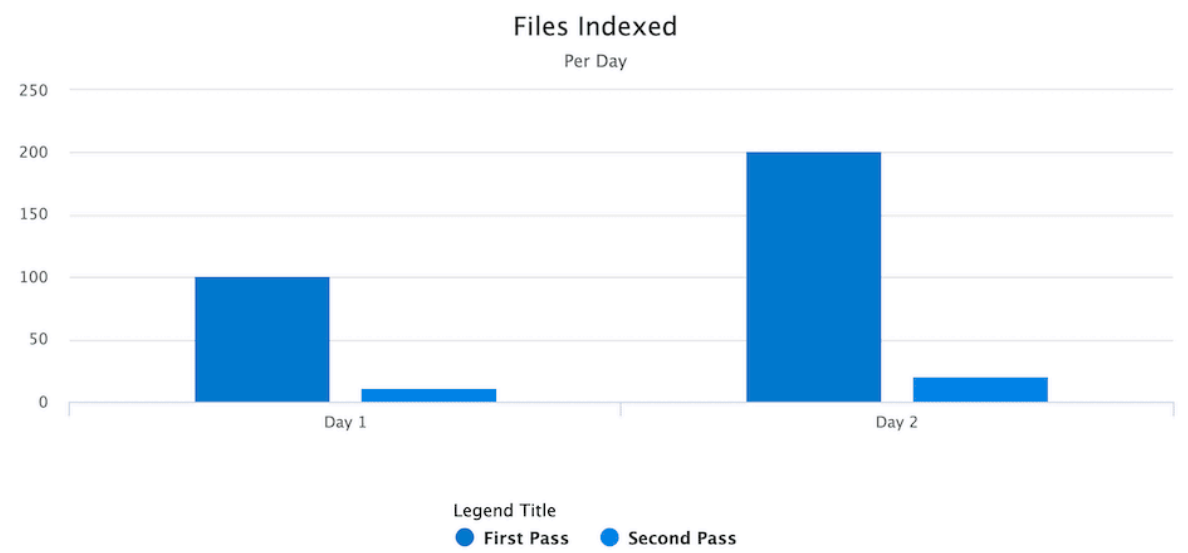



```
<chart:display response="response" title="Files Indexed" subtitle="Per Day">
    <series:facet name="test1" type="column" title="First Pass"></series:facet>
    <series:facet name="test2" type="column" title="Second Pass"></series:facet>
    <chart:legend title="Legend Title" align="right" vertical-align="top" layout="vertical"
```

```
y="100"></chart:legend>
</chart:display>
```

Legend on the Bottom

EXAMPLE



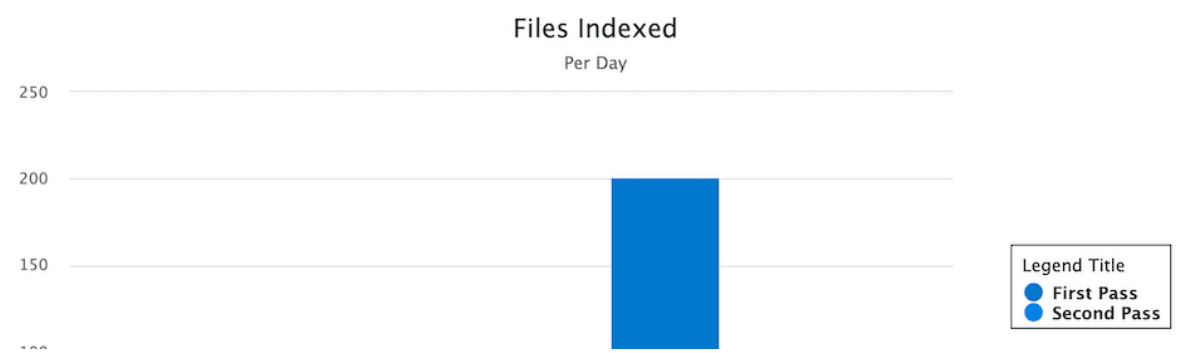

```
<chart:display response="response" title="Files Indexed" subtitle="Per Day">
    <series:facet name="test1" type="column" title="First Pass"></series:facet>
    <series:facet name="test2" type="column" title="Second Pass"></series:facet>
    <chart:legend title="Legend Title" align="center" vertical-align="bottom"
layout="vertical"></chart:legend>
</chart:display>
```

Adding a Border

Add an ID or class to the element wrapping your chart and configure the stroke width to add a border and the value rx to modify the border radius.

```
.chart-with-legend-border .highcharts-legend-box {
    fill: none;
    stroke-width: 1px;
    stroke: black;
    rx:2;
}
```

EXAMPLE



```
<chart:display response="response" title="Files Indexed" subtitle="Per Day">
    <series:facet name="test1" type="column" title="First Pass"></series:facet>
    <series:facet name="test2" type="column" title="Second Pass"></series:facet>
    <chart:legend title="Legend Title" align="right" vertical-align="middle"
layout="vertical"></chart:legend>
</chart:display>
```

chart:tagcloud

Description

The Chart Tag Cloud Directive

Usage

as element:

```
<chart:tagcloud response="{object}" facet-name="{string}" [title="{string}"]
[show="{number}"] [action="{string}"] [order="{string}"]> </chart:tagcloud>
```

Directive info

- This directive creates new scope.

Parameters

Param	Type	Details
response	**object**	A response object
facet-name	**string**	Facet name to use from the response object
title *(optional)*	**string**	The title that should be displayed above the tag cloud.
show *(optional)*	**number**	The maximum number of values to show in the tag cloud. Defaults to 15.
action *(optional)*	**string**	The target resource to which the tag cloud query should submit.
order *(optional)*	**string**	Key to order the data by, accepted values are `val`,`-val`,`-count`,`-weight`,`count` and `weight`.

chart:timeline

Description

The Timeline directive

Usage

as element:

```
<chart:timeline response="{object}" date="{string}" event="{string}" [milestone-grouping="{string}"] [orientation="{string}"] [max-width="{string}"] [max-height="{string}"] [query="{object}"] [platform="{string}"]> </chart:timeline>
```

Directive info

- This directive creates new scope.

Parameters

Param	Type	Details
response	**object**	A response object
date	**string**	The result field to use as a date for the timeline
event	**string**	The result field to use for the event content
milestoneGrouping *(optional)*	**string**	How to group the milestones i.e. year (yyyy), month (MM)
orientation *(optional)*	**string**	The orientation of the timeline. (horizontal or vertical. Default vertical)
maxWidth *(optional)*	**string**	Set max width of timeline container
maxHeight *(optional)*	**string**	Set max height of timeline container
query *(optional)*	**object**	Query object to fetch more results
platform *(optional)*	**string**	Platform to query for more results

chart:tooltip

Description

Tooltip Tag

Tooltips display information when you hover over a point or area of the chart, tooltips can be set across the whole chart or specifically to a series using the Chart Tooltip tag. This tag must be nested in the *Chart Display tag* or the series tags (*Series Facet Tag*, *Series Result List Tag* or *Series Time Tag*) if the tooltip is specific to a series.

Shared Tooltip

If you are rendering more then one series you can set the tooltip to be shared, this means it will display information about all the series for the point you've hovered over.

Usage

as element:

```
<chart:tooltip [header-format="{string}"] [point-format="{string}"] [shared="
{boolean}"]> </chart:tooltip>
```

Directive info

- This directive creates new scope.

Parameters

Param	Type	Details
header-format *(optional)*	**string**	Custom formatting of the data point tooltips header. Can make use of expressions '\{point.name}','\{point.x}' and '\{point.y}'. For example: '\{point.x} cm, \{point.y} kg'.
point-format *(optional)*	**string**	Custom formatting of the data point tooltips. Can make use of expressions '\{point.name}', '\{point.x}' and '\{point.y}'. For example: '\{point.x} cm, \{point.y} kg'.
shared *(optional)*	**boolean**	Whether the tooltip should show the point details of evert series (only work when placed within the Chart Display tag).

Example

Source

```
<div ng-controller="ExampleController"> <h3 id="example_tooltip-tag">Tooltip
Tag</h3> <p> When setting a tooltip you have access to two objects called <a
href="http://api.highcharts.com/highcharts/Point" target="_blank">Point</a> and <a
href="http://api.highcharts.com/highcharts/Series" target="_blank">Series</a>.
These can be used by wrapping them in '{}' eg. `{series.name} - {point.y}`. </p>
<tk-example> <chart:display response="response" title="Files Indexed" subtitle="Per
Day"> <series:facet name="test1" type="column" title="First Pass"></series:facet>
<series:facet name="test2" type="column" title="Second Pass"></series:facet>
<chart:tooltip point-format="{series.name}: <b>{point.y}</b><br/>"></chart:tooltip>
</chart:display> </tk-example> <tk-code-wrapper> <code hljs> <chart:display
response="response" title="Files Indexed" subtitle="Per Day"> <series:facet
name="test1" type="column" title="First Pass"></series:facet> <series:facet
name="test2" type="column" title="Second Pass"></series:facet> <chart:tooltip
point-format="{series.name}: <b>{point.y}</b><br/>"></chart:tooltip>
</chart:display> </code> </tk-code-wrapper> <hr/> <h3 id="example_shared-
tooltip">Shared Tooltip</h3> <p>If you are rendering more then one series you can
set the tooltip to be shared, this means it will display information about all the
series for the point you've hovered over.</p> <tk-example> <chart:display
response="response" title="Files Indexed" subtitle="Per Day"> <series:facet
name="test1" type="column" title="First Pass"></series:facet> <series:facet
name="test2" type="column" title="Second Pass"></series:facet> <chart:tooltip
point-format="{series.name}: <b>{point.y} </b><br/>" shared="true"></chart:tooltip>
</chart:display> </tk-example> <tk-code-wrapper> <code hljs> <chart:display
response="response" title="Files Indexed" subtitle="Per Day"> <series:facet
name="test1" type="column" title="First Pass"></series:facet> <series:facet
name="test2" type="column" title="Second Pass"></series:facet> <chart:tooltip
point-format="{series.name}: <b>{point.y} </b><br/>" shared="true"></chart:tooltip>
</chart:display> </code> </tk-code-wrapper> <hr/> <h3
id="example_percentages">Percentages</h3> <p>If you are rendering data within a pie
or donut chart you can user `{point.percentage} %` to show the percentage of the
segment you have selected in the chart.</p> <tk-example> <chart:display
response="response" title="Files Indexed"> <series:facet name="test1" type="pie"
title="First Pass"></series:facet> <chart:tooltip point-format="{series.name}: <b>
{point.percentage:.1f} %</b><br/>"></chart:tooltip> </chart:display> </tk-example>
<tk-code-wrapper> <code hljs> <chart:display response="response" title="Files
Indexed"> <series:facet name="test1" type="pie" title="First Pass"></series:facet>
<chart:tooltip point-format="{series.name}: <b>{point.percentage:.1f} %</b><br/>">
</chart:tooltip> </chart:display> </code> </tk-code-wrapper> <hr/> <h3
id="example_adding-colour">Adding Colour</h3> <p>You can add the colour of the
series to your highcharts tooltip by adding the following snippet:- <code>&lt;span
class='highcharts-color-{point.colorIndex}'&gt;●&lt;/span&gt;</code> To do this you
will need to use single quotes for your point-format eg. point-format='<span
class="my-class">Hello World</span>'. As the highcharts point format will not
render classes within double quotes. </p> <tk-example> <chart:display
response="response" title="Files Indexed"> <series:facet name="test1" type="pie"
title="First Pass"></series:facet> <chart:tooltip point-format='<span
class="highcharts-color-{point.colorIndex}">●</span> {series.name}: <b>
{point.percentage:.1f} %</b><br/>'></chart:tooltip> </chart:display> </tk-example>
<tk-code-wrapper> <code hljs hljs-no-escape> &lt;chart:display response="response"
title="Files Indexed"&gt; &lt;series:facet name="test1" type="pie" title="First
Pass"&gt;&lt;/series:facet&gt; &lt;chart:tooltip point-format='&lt;span
class=&quot;highcharts-color-{point.colorIndex}&quot;&gt;●&lt;/span&gt;
{series.name}: &lt;b&gt;{point.percentage:.1f}
%&lt;/b&gt;&lt;br/&gt;'&gt;&lt;/chart:tooltip&gt; &lt;/chart:display&gt; </code>
</tk-code-wrapper> <hr/> <h3 id="example_different-tooltips">Different
Tooltips</h3> <p>If for you need the tooltips to be different for each series you
can add the chart:tooltip tag inside your series to apply the tooltip to just that
series.</p> <tk-example> <chart:display response="response" title="Files Indexed"
subtitle="Per Day"> <series:facet name="test1" type="column" title="First Pass">
<chart:tooltip point-format='{series.name}: <b>{point.y} Files Indexed By Steve
</b><br/>'></chart:tooltip> </series:facet> <series:facet name="test2"
```

```
type="column" title="Second Pass"> <chart:tooltip point-format='{series.name}: <b>
{point.y} Files Indexed By Mike </b><br/>'></chart:tooltip> </series:facet>
</chart:display> </tk-example> <tk-code-wrapper> <code hljs> <chart:display
response="response" title="Files Indexed" subtitle="Per Day"> <series:facet
name="test1" type="column" title="First Pass"> <chart:tooltip point-
format='{series.name}: <b>{point.y} Files Indexed By Steve </b><br/>'>
</chart:tooltip> </series:facet> <series:facet name="test2" type="column"
title="Second Pass"> <chart:tooltip point-format='{series.name}: <b>{point.y} Files
Indexed By Mike </b><br/>'></chart:tooltip> </series:facet> </chart:display>
</code> </tk-code-wrapper> </div>
```

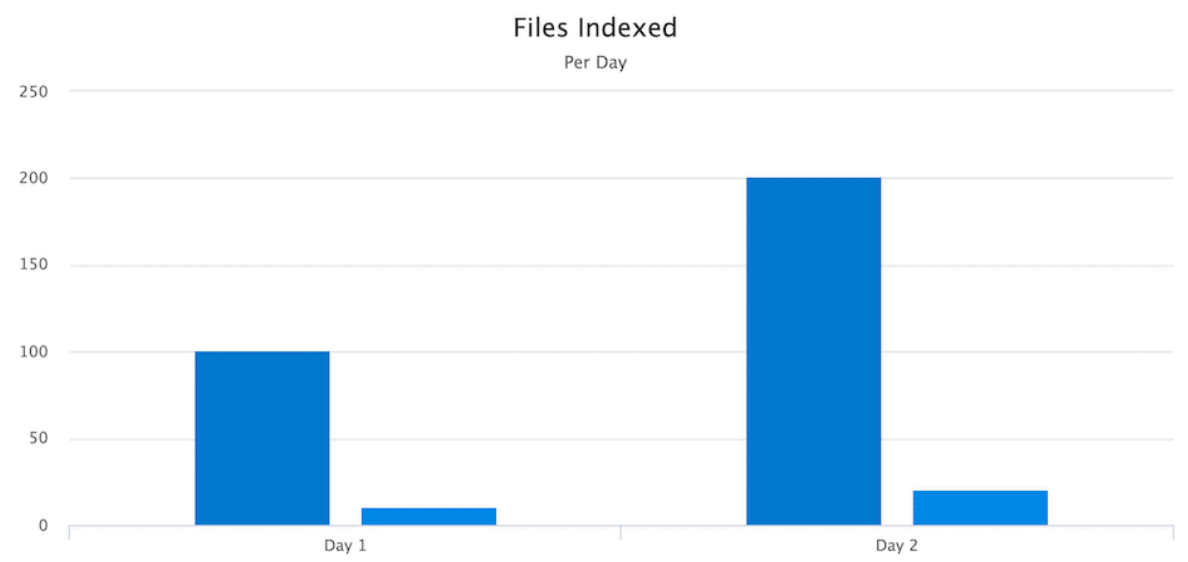
```
angular.module('lightning') .controller('ExampleController',
['$scope','$timeout','ResponseService', function($scope,$timeout,ResponseService) {
$scope.response = { page: 2, query: { rpp: 50 }, facets: { test1: { filters: [ {
val: { dsp: 'Day 1', act: 'Day 1', }, count: 100, }, { val: { dsp: 'Day 2', act:
'Day 2', }, count: 200, } ] }, test2: { filters: [ { val: { dsp: 'Day 1', act: 'Day
1', }, count: 10, }, { val: { dsp: 'Day 2', act: 'Day 2', }, count: 20, } ] }, },
results: [ { result: { fields: { label: 'label1', value: 1 } }, }, { result: {
fields: { label: 'label2', value: 2 } }, hits: { act: ['Hits', 264] } }] }
ResponseService.setResponse('response',$scope.response); }]);
```

Result

Tooltip Tag

When setting a tooltip you have access to two objects called Point and Series. These can be used by wrapping them in '{}' eg. `{series.name} - {point.y}`.

EXAMPLE

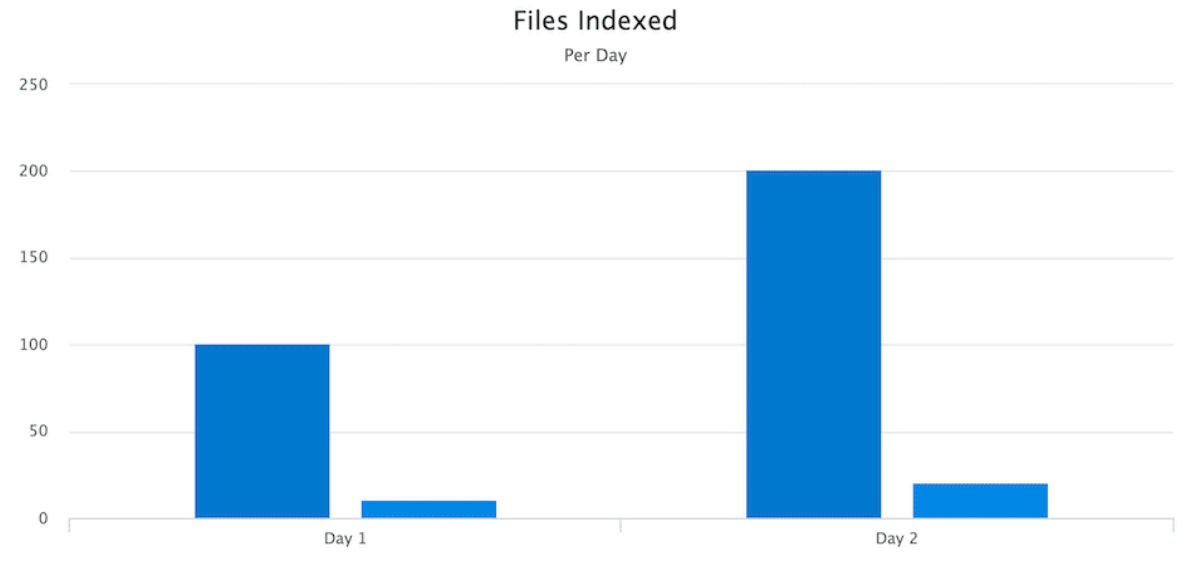



```
<chart:display response="response" title="Files Indexed" subtitle="Per Day">
    <series:facet name="test1" type="column" title="First Pass"></series:facet>
    <series:facet name="test2" type="column" title="Second Pass"></series:facet>
    <chart:tooltip point-format="{series.name}: <b>{point.y}</b><br/>"></chart:tooltip>
</chart:display>
```

Shared Tooltip

If you are rendering more then one series you can set the tooltip to be shared, this means it will display information about all the series for the point you've hovered over.

EXAMPLE

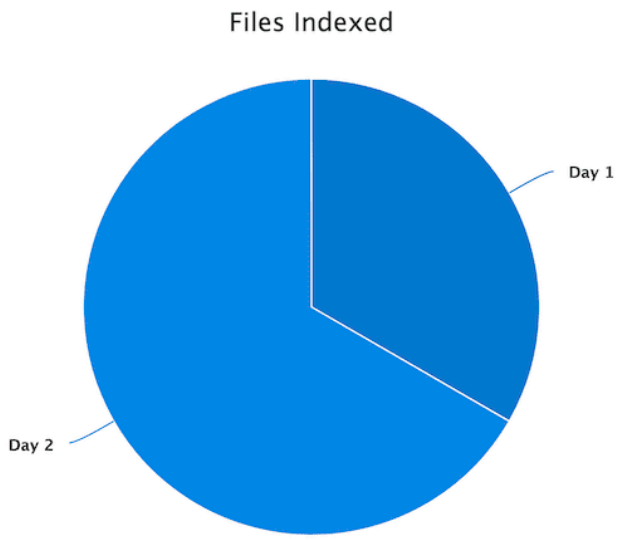


```
<chart:display response="response" title="Files Indexed" subtitle="Per Day">
    <series:facet name="test1" type="column" title="First Pass"></series:facet>
    <series:facet name="test2" type="column" title="Second Pass"></series:facet>
    <chart:tooltip point-format="{series.name}: <b>{point.y} </b><br/>" shared="true">
</chart:tooltip>
</chart:display>
```

Percentages

If you are rendering data within a pie or donut chart you can user `{point.percentage} %` to show the percentage of the segment you have selected in the chart.

EXAMPLE

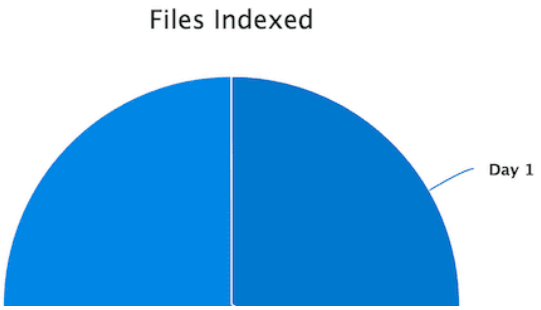


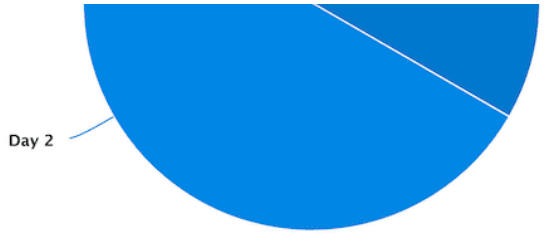

```
<chart:display response="response" title="Files Indexed">
    <series:facet name="test1" type="pie" title="First Pass"></series:facet>
    <chart:tooltip point-format="{series.name}: <b>{point.percentage:.1f} %</b><br/>">
</chart:tooltip>
</chart:display>
```

Adding Colour

You can add the colour of the series to your highcharts tooltip by adding the following snippet:- `<span class='highcharts-color-{point.colorIndex}'>●</span>` To do this you will need to use single quotes for your point-format eg. point-format='Hello World'. As the highcharts point format will not render classes within double quotes.

EXAMPLE



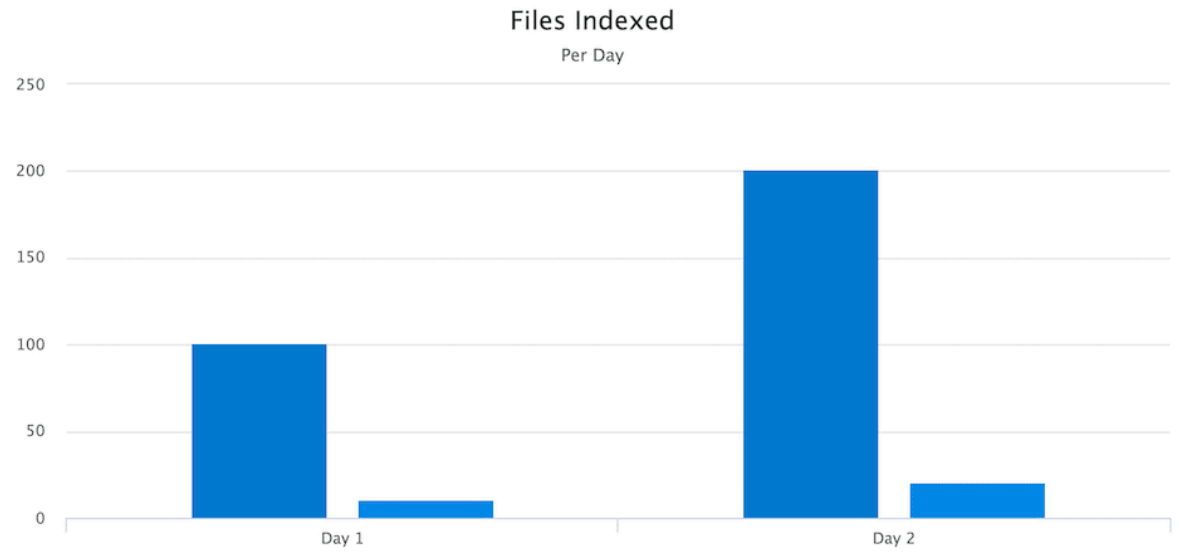



```
<chart:display response="response" title="Files Indexed">
    <series:facet name="test1" type="pie" title="First Pass"></series:facet>
    <chart:tooltip point-format='<span class="highcharts-color-{point.colorIndex}">●</span>
{series.name}: <b>{point.percentage:.1f} %</b><br/>'></chart:tooltip>
</chart:display>
```

Different Tooltips

If for you need the tooltips to be different for each series you can add the chart:tooltip tag inside your series to apply the tooltip to just that series.

EXAMPLE




```
<chart:display response="response" title="Files Indexed" subtitle="Per Day">
    <series:facet name="test1" type="column" title="First Pass">
        <chart:tooltip point-format="{series.name}: <b>{point.y} Files Indexed By Steve </b>
<br/>"></chart:tooltip>
    </series:facet>
    <series:facet name="test2" type="column" title="Second Pass">
        <chart:tooltip point-format="{series.name}: <b>{point.y} Files Indexed By Mike </b>
<br/>"></chart:tooltip>
    </series:facet>
</chart:display>
```

class:toggle

Description

Applies a class to the target when toggled on.

Usage

as element:

```
<class:toggle target="{string}" toggle-class="{string}" [name="{string}"]>
</class:toggle>
```

Directive info

- This directive creates new scope.

Parameters

Param	Type	Details
target	**string**	jQuery selector for element
toggle-class	**string**	Name of class to toggle
name *(optional)*	**string**	Name of toggle. If set the user's preference for the toggle will be saved

Example

```
<search:class-toggle target="#chart" class="hide" name="chart">Toggle</search:class-
toggle>
```

click:outside

Description

Calls function when a click event occurs outside of the element. Good for closing side panels, menus, etc.

Usage

as attribute

```
<ANY click-outside="{expression}" [outside-if-not="{string}"]> ... </ANY>
```

Parameters

Param	Type	Details
click-outside	**expression**	Function to call when element is clicked outside of.
outside-if-not *(optional)*	**string**	Selector to match any elements that should not trigger the click-outside function

collaborate:bookmark

Description

Creates a new bookmark.

- If used within a result you will need to add the following workflow processor to your platform in order to get the status of the bookmark and the number of people who have bookmarked the result

```
workflow:
 twigkit.social.processor.AddBookmarkFactsProcessor
```

Usage

as element:

```
<collaborate:bookmark target="{string}" title="{string}" [topic="{number}"]
[collection="{string}"] [url="{string}"] [enable-tooltip="{boolean}"]>
</collaborate:bookmark>
```

Parameters

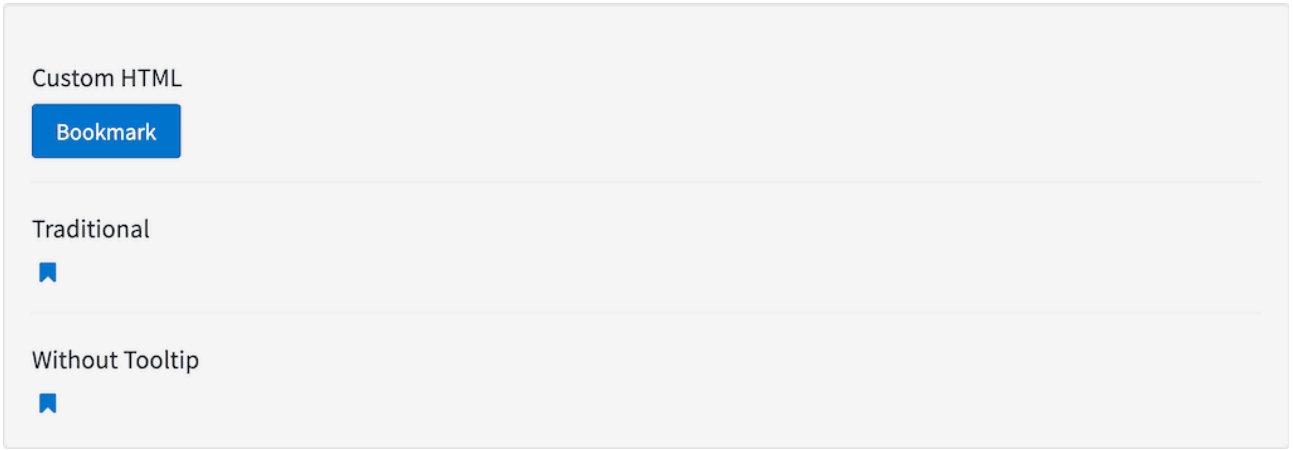
Param	Type	Details
target	**string**	A unique identifier for a document that is being bookmarked. Example values: document123 Maximum length: 2083 characters
title	**string**	A title for the new bookmark. Example values: Interesting document Maximum length: 4000 characters
topic *(optional)*	**number**	A topic to which the new bookmark should belong.
collection *(optional)*	**string**	The collection to which the target document belongs.
url *(optional)*	**string**	A URL which the new bookmark should link to. Default: the bookmark target, if it is a URL
enable-tooltip *(optional)*	**boolean**	Whether to enable the tooltip (Default:true).

Example

Source

```
<h3 id="example_custom-html">Custom HTML</h3> <collaborate:bookmark><a
styling="button">Bookmark</a></collaborate:bookmark> <hr/> <h3
id="example_traditional">Traditional</h3> <collaborate:bookmark>
</collaborate:bookmark> <hr/> <h3 id="example_without-tooltip">Without Tooltip</h3>
<collaborate:bookmark enable-tooltip="false"></collaborate:bookmark>
```

Result



collaborate:bookmark-list

Description

The bookmark list directive to render a list of bookmarks, you can put your own html inside the bookmark list to customise it, in order to do this you will need to do the following :-

```
<collaborate:bookmark-list>
     <li ng-repeat="bookmark in $parent.bookmarks">YOUR CUSTOM CODE HERE</li>
</collaborate:bookmark-list>
```

Usage

as element:

```
<collaborate:bookmark-list [title="{string}"] [delete-label="{string}"] [target="
{string}"] [topic="{number}"] [collection="{string}"] [date-format="{string}"]
[url-target="{string}"]> </collaborate:bookmark-list>
```

Directive info

- This directive creates new scope.

Parameters

Param	Type	Details
title *(optional)*	**string**	If set, title will be used displayed above bookmarks
delete-label *(optional)*	**string**	Label to display for the delete action. Default: “x”
target *(optional)*	**string**	If set, only display bookmarks related to this target document.
topic *(optional)*	**number**	Optionally, bookmarks can be scoped to a particular topic. If this attribute is set, then the tag will list only those bookmarks that are linked to that topic. Passing topic 0 will check that a topic has not be associated with the bookmark
collection *(optional)*	**string**	Optionally, bookmarks can be scoped to a particular collection, identified by a string ID. If this attribute is set, then the tag will list only those bookmarks that belong to the given collection.
date-format *(optional)*	**string**	Format to use for the bookmark created date. Default: `dd/MM/yyyy h:mma`. See #/api/lightning.filter:dateFormat[date format filter] for valid formats.
url-target *(optional)*	**string**	Set an HTML target attribute for the anchor tag. For example use _blank to have the bookmark link open in a new window.

collaborate:comment-form

Description

The comment form directive to render a comment form

Usage

as element:

```
<collaborate:comment-form target="{string}" platform="{object}" [collection="
{string}"] [topic="{number}"]> </collaborate:comment-form>
```

Directive info

- This directive creates new scope.

Parameters

Param	Type	Details
target	**string**	The ID of the document that should be commented on.
platform	**object**	The platform Object
collection *(optional)*	**string**	Collection the comment belongs to.
topic *(optional)*	**number**	Topic the comment belongs to.

collaborate:comment-list

Description

Comment list component for displaying a list of comments.

If used within a result you will need to add the following workflow processor to your platform in order to get the comments that have been made on this result.

```
workflow:
    twigkit.social.processor.AddCommentFactsProcessor
```

Usage

as element:

```
<collaborate:comment-list target="{string}" platform="{object}" [collection="
{string}"] [topic="{number}"] [format="{string}"] [show="{number}"] [show-more="
{number}"] [empty-text="{string}"] [avatars="{string}"]> </collaborate:comment-
list>
```

Directive info

- This directive creates new scope.

Parameters

Param	Type	Details
target	**string**	The ID of the document that should be commented on.
platform	**object**	The platform Object
collection *(optional)*	**string**	If set, only display comments restricted to the given collection.
topic *(optional)*	**number**	If set, only display comments restricted to the given type.
format *(optional)*	**string**	How to format the date (formats supported: https://docs.angularjs.org/api/ng/filter/date *(https://docs.angularjs.org/api/ng/filter/date)* and 'relative'which outputs the relative date (e.g. 2 hours ago)). Default: relative
show *(optional)*	**number**	How many comments should be displayed initially (default:2).
show-more *(optional)*	**number**	How many more comments should be added to the list each time the user clicks 'Show more' (default: 4).
empty-text *(optional)*	**string**	Message to display when there are no comments to show (default: "").
avatars *(optional)*	**string**	Whether to display avatars next to each comment (default: "true").

collaborate:like

Description

Creates a like button.

If used within a result you will need to add the following workflow processor to your platform configuration file in order to get the status of the like and the number of people who have liked the result. The default file is `platforms/fusion/data.conf`.

```
workflow:
    twigkit.social.processor.AddLikeFactsProcessor
```

Usage

as element:

```
<collaborate:like target="{string}" [topic="{number}"] platform="{string}"
[collection="{string}"] [enable-tooltip="{boolean}"]> </collaborate:like>
```

Parameters

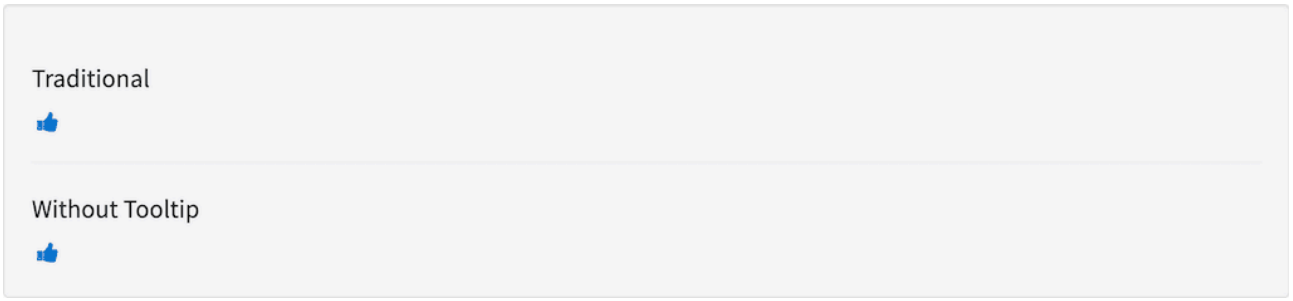
Param	Type	Details
target	**string**	A unique identifier for a document that is being liked. Example values: document123 Maximum length: 2083 characters
topic *(optional)*	**number**	A topic to which the new like should belong.
platform	**string**	Name of the platform to send the data to.
collection *(optional)*	**string**	The collection to which the target document belongs.
enable-tooltip *(optional)*	**boolean**	Whether to enable the tooltip (Default:true).

Example

Source

```
<h3 id="example_traditional">Traditional</h3> <collaborate:like></collaborate:like>
<hr/> <h3 id="example_without-tooltip">Without Tooltip</h3> <collaborate:like
enable-tooltip="false"></collaborate:like>
```

Result



collaborate:saved-query-list

Description

The saved query list directive to render a list of saved queries

Usage

as element:

```
<collaborate:saved-query-list user="{object}" [title="{string}"] [allow-saving="{string}"] [date-format="{string}"]> </collaborate:saved-query-list>
```

Directive info

- This directive creates new scope.

Parameters

Param	Type	Details
user	**object**	Display saved queries by the given user (type: twigkit.social.model.Profile).
title *(optional)*	**string**	If set, title will be used displayed above bookmarks
allow-saving *(optional)*	**string**	If set to false the save search input will not be shown
date-format *(optional)*	**string**	Format to use for the saved query created date. Default: `dd/MM/yyyy h:mma`. See #/api/lightning.filter:dateFormat[date format filter] for valid formats.

Events

savedQueryAdded

When a query is saved this event broadcast with the name of the query being saved, the URL being used and the topic if used.

```
angular.module('twigkitLightApp') .controller('MainCtrl', ['$scope', function ($scope) { $scope.$on('savedQueryAdded',function(query){ alert('Query – ' + query.name + ' have been saved.'); }); }]);
```

Type:

broadcast

Target:

root scope

savedQueryDeleted

When a query is deleted this event is broadcast so that any custom alerts or functions can be run afterwards like the example below.

```
angular.module('twigkitLightApp') .controller('MainCtrl', ['$scope', function ($scope) { $scope.$on('savedQueryDeleted',function(){ alert('Query has been deleted.'); }); }]);
```

Type:

broadcast

Target:

root scope

collaborate:topic-form

Description

Generates a form you can use to create a topic.

Usage

as element:

```
<collaborate:topic-form [name="{string}"] [button-label="{string}"] [namespace="{string}"] [placeholder="{string}"] [show-description="{boolean}"] [description-placeholder="{string}"]> </collaborate:topic-form>
```

Directive info

This directive creates new scope.

Parameters

Param	Type	Details
name *(optional)*	string	Default title for a topic.
button-label *(optional)*	string	Label of form submit button. *(default: Save)*
namespace *(optional)*	string	Determines whether the topics created using the form will be "local" (restricted to the user) or "global". *(default: local)*
placeholder *(optional)*	string	Placeholder value to put in the topic name text box. *(default: Type a name for the new topic…)*
show-description *(optional)*	string	Whether to include a text box for adding a description for the topic.
description-placeholder *(optional)*	string	Placeholder value to put in the description text box *(default: Description)*

collaborate:topic-list

Description

The topic list directive is used to render a list of topics specific to your organization. To customize the topic list, enter your custom html code with the following commands:

```
<collaborate:topic-list> <li ng-repeat="topic in $parent.topics">YOUR CUSTOM CODE HERE</li> </collaborate:topic-list>
```

Usage

as element:

```
<collaborate:topic-list [title="{string}"] [delete-label="{string}"] [show="{number}"]> </collaborate:topic-list>
```

Directive information

This directive creates new scope.

content:boolean

Description

Creates a checkbox boolean

Usage

as element:

```
<content:boolean label="{string}" field="{string}" [true-value="{string}"] [false-value="{string}"]> </content:boolean>
```

Parameters

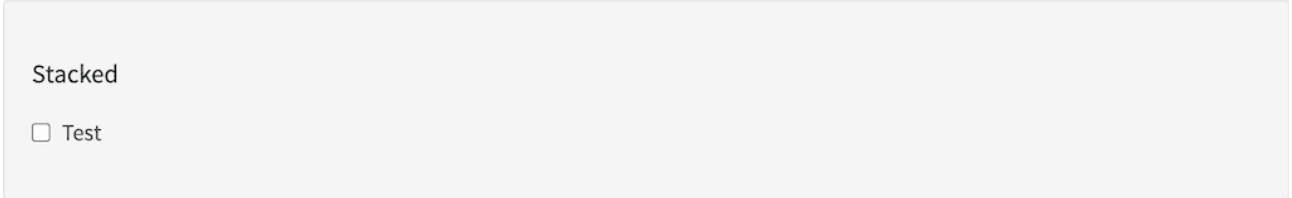
Param	Type	Details
label	**string**	The label for the textarea
field	**string**	The name of the field.
true-value *(optional)*	**string**	The value to use when true, (default: true)
false-value *(optional)*	**string**	The value use when false, (default: false)

Example

Source

```
<h3 id="example_stacked">Stacked</h3> <content:form styling="form-stacked">
<content:boolean label="Test" field="test"></content:boolean> </content:form>
```

Result



content:form

Description

Creates a form, all inputs into the form are attached to a variable called model.

Adding a platform to the form will send the request to the Platform Store API *(http://dev.twigkit.net/docs/modules/Webservices/Store/)* and create the model as a new result .

Usage

as element:

```
<content:form [platform="{string}"] [legend="{string}"] [action="{string}"]
[content="{Object}"]> </content:form>
```

Directive info

- This directive creates new scope.

Parameters

Param	Type	Details
platform *(optional)*	**string**	The platform to post the data to.
legend *(optional)*	**string**	Text to display in the legend of the form.
action *(optional)*	**string**	Whether the form should be creating a new record (`store`) or updating an existing record (`update`). Default:`store`, options `store` or `update`
content *(optional)*	**Object**	Key value pair object or result to map the form data to.

Events

contentFormPostError

When the content form is posted to the API unsuccessfully this event will be broadcasted with the error received from the API.

Type:

broadcast

Target:

root scope

contentFormPostSuccess

When the content form is posted to the API successfully this event will be broadcasted with the model sent to the API.

Type:

broadcast

Target:

root scope

Example

Source

```
<div ng-controller="ExampleController"> <search:platform var="platform"
conf="platform.test"></search:platform> <layout:grid> <layout:block md="1-1"> <h3
id="example_styling">Styling</h3> </layout:block> <layout:block md="1-2"> <h4
id="example_styling">(form-aligned)</h4> <content:form styling="form-aligned"
platform="platform"> <content:input label="id" field="id" value="123456"
hidden="true"></content:input> <content:input label="Name" field="name"
required="true"></content:input> <content:textarea label="Address" field="Address"
required="true"></content:textarea> <content:submit></content:submit>
</content:form> </layout:block> <layout:block md="1-2"> <h4 id="example_styling">
(form-stacked)</h4> <content:form styling="form-stacked" content="model1"
platform="platform"> <content:input label="id" field="id" value="12345"
hidden="true"></content:input> <content:input label="Name" field="name"
required="true"></content:input> <content:textarea label="Address" field="Address">
</content:textarea> <content:submit></content:submit> </content:form>
</layout:block> </layout:grid> <hr/> <h3 id="example_attributes">Attributes</h3>
<h4 id="example_attributes_legend">Legend</h4> <content:form legend="Please
complete the form." content="result"> <content:input label="id" field="id"
value="1234" hidden="true"></content:input> <content:input label="Name"
field="name" required="true"></content:input> <content:textarea label="Address"
field="Address"></content:textarea> <content:submit></content:submit>
</content:form> <hr/> <h4 id="example_attributes_content">Content</h4> <button
styling="button" ng-click="changeModel()">Update Content Model</button>
<layout:grid> <layout:block md="1-1"> <p>Click update content model button to
update the values in the inputs.</p> <content:form styling="form-stacked"
content="model1" platform="platform"> <strong>Form Model</strong> <pre>
{{$parent.model}}</pre> <content:input label="id" field="id" value="12345"
hidden="true"></content:input> <content:input label="Name" field="name"
required="true"></content:input> <content:textarea label="Address" field="Address">
</content:textarea> <content:submit></content:submit> </content:form>
</layout:block> <layout:block md="1-1"> <p>To only initiated the form with the
values from a result or scope object use <code>content="::myModelObject"</code>. If
you click update content model the values of this form will not change. (see <a
href="https://docs.angularjs.org/guide/expression#one-time-binding"
target="_blank">one time bindings</a>)</p> <content:form styling="form-stacked"
content="::model1" platform="platform"> <strong>Form Model</strong> <pre>
{{$parent.model}}</pre> <content:input label="id" field="id" value="12345"
hidden="true"></content:input> <content:input label="Name" field="name"
required="true"></content:input> <content:textarea label="Address" field="Address">
</content:textarea> <content:submit></content:submit> </content:form>
</layout:block> </layout:grid> </div>
```

```
angular.module('lightning') .controller('ExampleController', ['$scope', function($scope) { $scope.model1 = { name:'foo', Address:'bar' } function generateRandomString(size){ var text = ""; var possible = "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789"; for( var i=0; i < size; i++ ) text += possible.charAt(Math.floor(Math.random() * possible.length)); return text; } $scope.changeModel = function(){ $scope.model1 = { name:generateRandomString(4), Address:generateRandomString(3) } }; $scope.result ={ result: { fields: { name:{ val:'George', } , Address:{ val:'123 George Way', }, } }, } }]);
```

Result

Styling

(form-aligned)

Name *

Address *

Submit

(form-stacked)

Name *

foo

Address

bar

Submit

Attributes

Legend

Please complete the form.

Name *

George

Address

123 George Way

Submit

Content

Update Content Model

Click update content model button to update the values in the inputs.

Form Model

```
{"id":"12345","name":"foo","Address":"bar"}
```

Name *

foo

Address

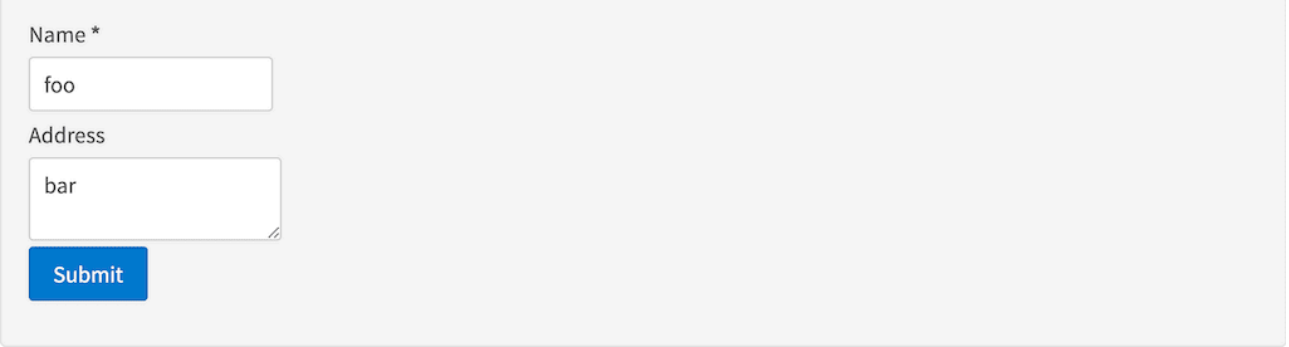
bar

Submit

To only initiated the form with the values from a result or scope object use `content="::myModelObject"`. If you click update content model the values of this form will not change. (see one time bindings)

Form Model

```
{"id":"12345","name":"foo","Address":"bar"}
```



content:input

Description

Creates a textbox

Usage

as element:

```
<content:input label="{string}" field="{string}" [required] [value="{string}"]
[hidden="{boolean}"] [placeholder="{string}"]> </content:input>
```

Parameters

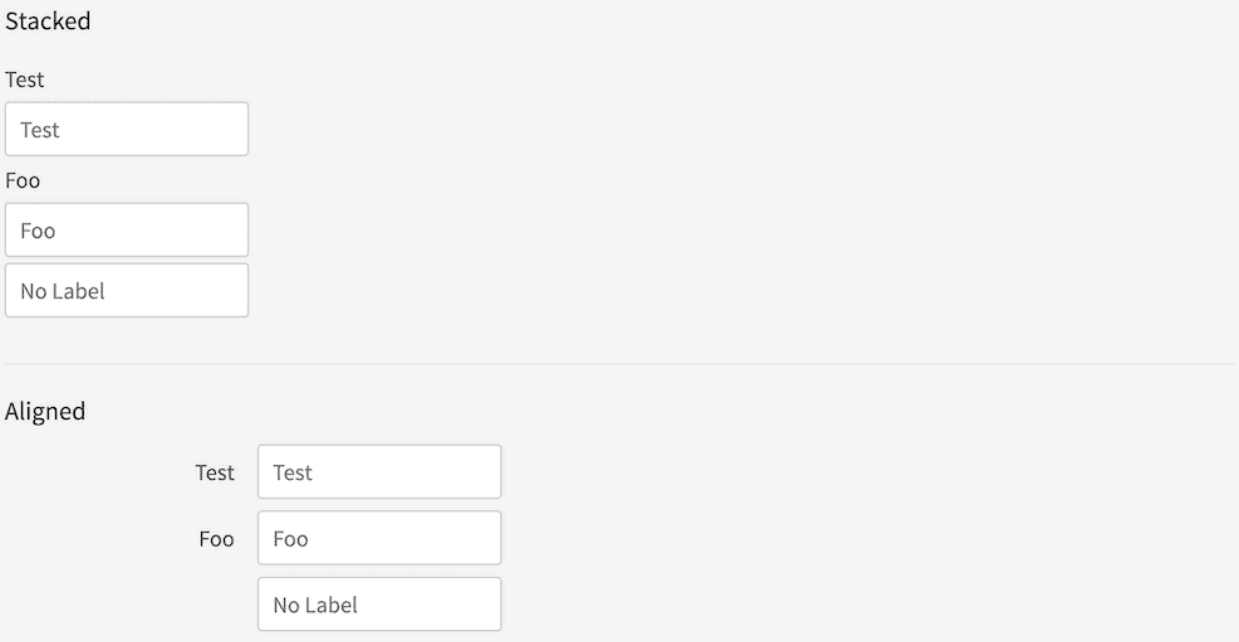
Param	Type	Details
label	**string**	The label for the textbox
field	**string**	The name of the field.
required *(optional)*	**boolean**	Whether the field is required not, if it is a warning will be displayed until the field is value. Default:false.
value *(optional)*	**string**	Value to display in the textbox.
hidden *(optional)*	**boolean**	Whether the textbox should be displayed.
placeholder *(optional)*	**string**	The placeholder for the textbox, if no placeholder is defined the label will be used as the placeholder.

Example

Source

```
<h3 id="example_stacked">Stacked</h3> <content:form styling="form-stacked">
<content:input label="Test" field="test"></content:input> <content:input
label="Foo" field="bar"></content:input> <content:input label="" field="barfoo"
placeholder="No Label"></content:input> </content:form> <hr/> <h3
id="example_aligned">Aligned</h3> <content:form styling="form-aligned">
<content:input label="Test" field="test"></content:input> <content:input
label="Foo" field="bar"></content:input> <content:input label="" field="barfoo"
placeholder="No Label"></content:input> </content:form>
```

Result



content:radio

Description

Creates a radio button

Usage

as element:

```
<content:radio label="{string}" field="{string}" value="{string}">
</content:radio>
```

Parameters

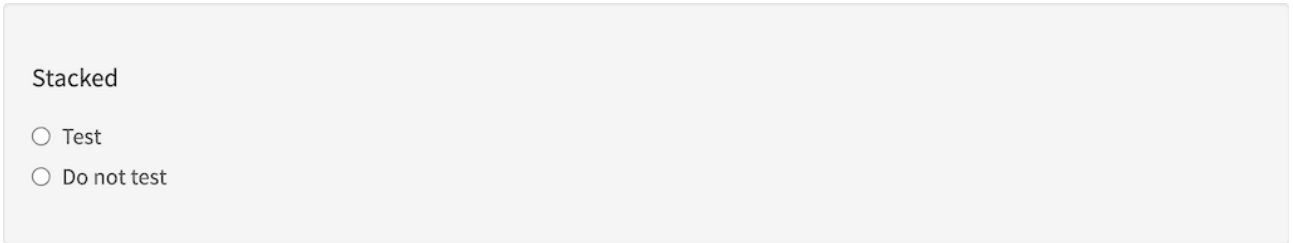
Param	Type	Details
label	**string**	The label for the textarea
field	**string**	The name of the field.
value	**string**	The value of the field when selected

Example

Source

```
<h3 id="example_stacked">Stacked</h3> <content:form styling="form-stacked">
<content:radio label="Test" field="test" value="tested"></content:radio>
<content:radio label="Do not test" field="test" value="untested"></content:radio>
</content:form>
```

Result



content:submit

Description

Creates a submit button

Usage

as element:

```
<content:submit> </content:submit>
```

content:textarea

Description

Creates a textbox

Usage

as element:

```
<content:textarea label="{string}" field="{string}" [required] [value="{string}"]
[placeholder="{string}"]> </content:textarea>
```

Parameters

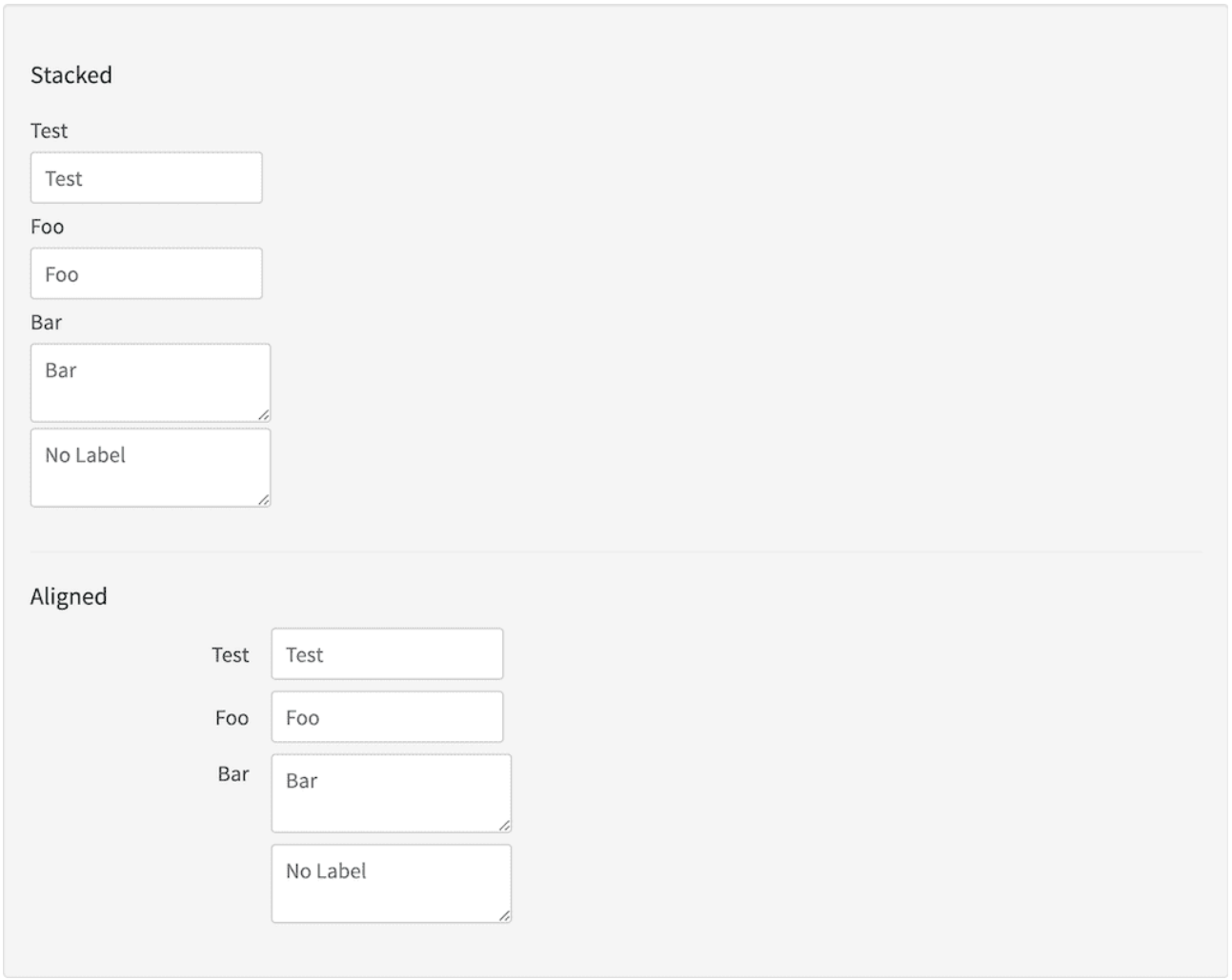
Param	Type	Details
label	**string**	The label for the textarea
field	**string**	The name of the field.
required *(optional)*	**boolean**	Whether the field is required not, if it is a warning will be displayed until the field is value. Default:false.
value *(optional)*	**string**	Value to display in the textarea.
placeholder *(optional)*	**string**	The placeholder for the textarea, if no placeholder is defined the label will be used as the placeholder.

Example

Source

```
<h3 id="example_stacked">Stacked</h3> <content:form styling="form-stacked">
<content:input label="Test" field="test"></content:input> <content:input
label="Foo" field="bar"></content:input> <content:textarea label="Bar" field="foo">
</content:textarea> <content:textarea label="" field="foobar" placeholder="No
Label"></content:textarea> </content:form> <hr/> <h3
id="example_aligned">Aligned</h3> <content:form styling="form-aligned">
<content:input label="Test" field="test"></content:input> <content:input
label="Foo" field="bar"></content:input> <content:textarea label="Bar" field="foo">
</content:textarea> <content:textarea label="" field="foobar" placeholder="No
Label"></content:textarea> </content:form>
```

Result


Stacked
Test
Test
Foo
Foo
Bar
Bar
No Label
Aligned
Test
Test
Foo
Foo
Bar
Bar
No Label

controller

Description

Wraps the html in a controller.

Usage

as attribute

```
<ANY controller="{string}"> ... </ANY>
```

Parameters

Param	Type	Details
controller	**string**	The name of the controller.

Example

Source

```
<div controller="ExampleController"> <p>{{foo}} <a href="javascript:void(0);" ng-
click="alert()" styling="button">Alert!</a> </p> </div> <hr/> <div
controller="Example2Controller"> <p>{{bar}}</p> </div>
```

```
angular.module('lightning') .controller('ExampleController', ['$scope','$timeout',
function($scope,$timeout) { $scope.foo = 'bar'; $scope.alert = function(){
alert('Hello World!') }; }]) .controller('Example2Controller',
['$scope','$timeout', function($scope,$timeout) { $scope.bar = 'foo'; }]);
```

Result

foo

default:src

Description

When an img tag gets a 404 the src attribute will be replaced with the default-src attribute

Usage

as attribute

```
<ANY default-src> ... </ANY>
```

dev:track-digests

Description

Prints out the number of digests that have taken place. https://toddmotto.com/super-fast-angular-ng-model-options-limit-digest-cycles/ *(https://toddmotto.com/super-fast-angular-ng-model-options-limit-digest-cycles/)*

Usage

as element:

```
<dev:track-digests> </dev:track-digests>
```

Example

```
<dev:track-digests></dev:track-digests>
```

drawer

Description

Adds a class for the drawer

Usage

as attribute

```
<ANY drawer="{number}"> ... </ANY>
```

Parameters

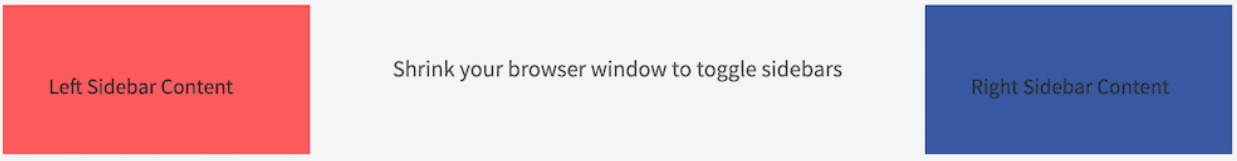
Param	Type	Details
drawer	**string**	The direction the drawer should come from (left or right);

Example

Source

```
<layout:grid> <layout:block tiny="1-4"> <layout:box> <search:offcanvas-toggle
target="#left-sidebar"></search:offcanvas-toggle> </layout:box> </layout:block>
<layout:block tiny="1-2"> </layout:block> <layout:block tiny="1-4"> <layout:box>
<search:offcanvas-toggle target="#right-sidebar" style="float:right;">
</search:offcanvas-toggle> </layout:box> </layout:block> </layout:grid>
<layout:grid style="row-offcanvas"> <layout:block medium="1-4" drawer="left"
id="left-sidebar" > <layout:box class="tk-stl-bg-red" > <layout:sidebar> <h4
id="example_left-sidebar-content">Left Sidebar Content</h4> </layout:sidebar>
</layout:box> </layout:block> <layout:block medium="1-2"> <layout:box> <p
style="text-align:center;">Shrink your browser window to toggle sidebars</p>
</layout:box> </layout:block> <layout:block medium="1-4" drawer="right" id="right-
sidebar" > <layout:box class="tk-stl-bg-blue" > <layout:sidebar> <h4
id="example_right-sidebar-content">Right Sidebar Content</h4> </layout:sidebar>
</layout:box> </layout:block> </layout:grid>
```

Result



dynamic:styling

Description

Works the same as the styling directive but watches for updates. This should only be used if it is required that the styling classes are updated after their initial value to improve performance.

Usage

as attribute

```
<ANY dynamic-styling="{string}"> ... </ANY>
```

Parameters

Param	Type	Details
dynamicStyling	**string**	separated string of styles to apply to the element

Example

Source

```
<layout:box dynamic-styling="{{styles}}"> <button styling="button" ng-
click="toggleStyles()">Toggle box styling</button> </layout:box>
```

```
angular.module('lightning') roller('ExampleController', ['$scope', function($scope)
{ $scope.styles = 'bg-raspberry text-raspberry-dark'; $scope.toggleStyles =
function () { if ($scope.styles == 'bg-raspberry text-raspberry-dark') {
$scope.styles = 'bg-sunflower text-sunflower-dark'; } else { $scope.styles = 'bg-
raspberry text-raspberry-dark'; } }; }]);
```

Result

Toggle box styling

esri:heatmap

Description

The heatmap directive

Usage

as element:

```
<esri:heatmap response="{string}" facet-name="{string}" [height="{string}"]
[width="{string}"] [basemap="{string}"]> </esri:heatmap>
```

Directive info

- This directive creates new scope.

Parameters

Param	Type	Details
response	**string**	Name of the response.
facet-name	**string**	Facet containing the data.
height *(optional)*	**string**	The height of the map container. Default is 400px.
width *(optional)*	**string**	The width of the map container. Default is 100%.
basemap *(optional)*	**string**	The map to use to display the points on. Default: topo - Esri options *(https://developers.arcgis.com/javascript/4.2/api-reference/esri-Map.html#basemap)*

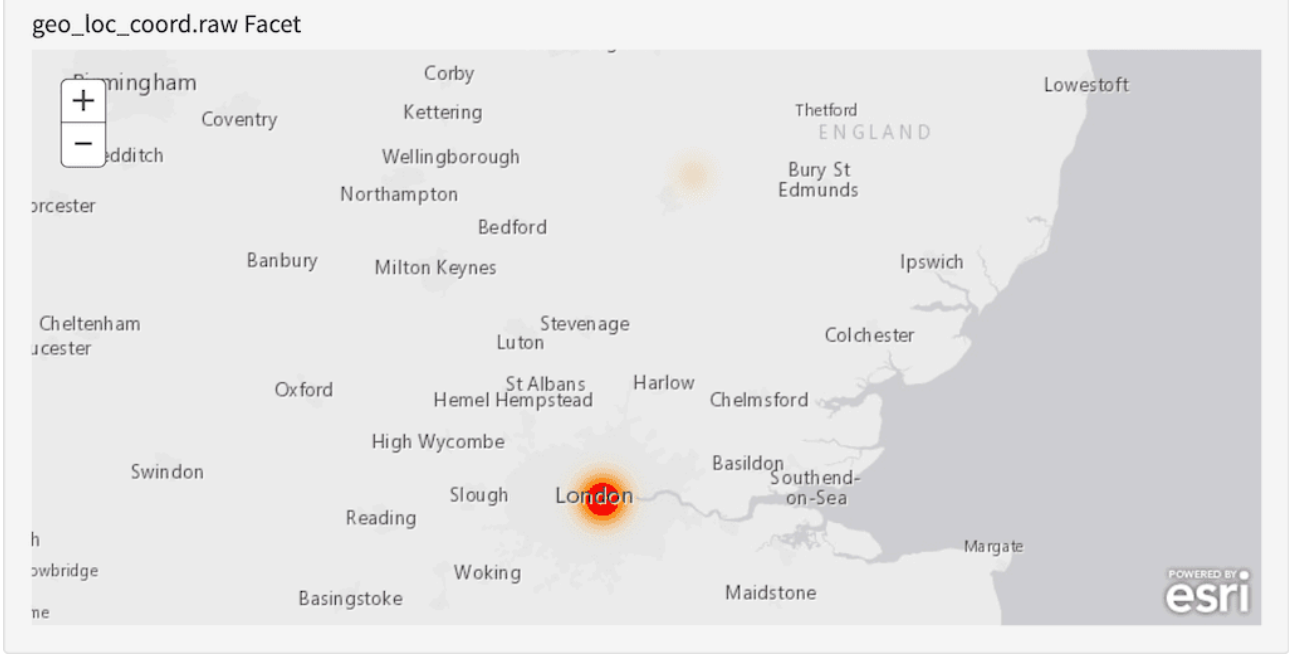
Example

Source

```
<h3 id="example_geo_loc_coordraw-facet">geo_loc_coord.raw Facet</h3> <esri:heatmap
response="response" facet-name="geo_loc_coord.raw" basemap="gray"></esri:heatmap>
```

```
angular.module('lightning') .controller('ExampleController', ['$scope','$timeout',
function($scope,$timeout) { $scope.response = { page: 2, query: { rpp: 50 },
facets: { heat: { filters: [ { val: '37.770090, -122.408904', count: 1, }, { val:
'37.782745, -122.444586', count: 1, }, { val: '37.782842, -122.443688', count: 1,
}, { val: '37.782919, -122.442815', count: 1, }, { val: '37.782992, -122.442112',
count: 1, }, { val: '37.783100, -122.441461', count: 1, }, { val: '37.783206,
-122.440829', count: 1, }, { val: '37.783273, -122.440324', count: 1, }, { val:
'37.783316, -122.440023', count: 1, }, { val: '37.783357, -122.439794', count: 1,
}, { val: '37.783371, -122.439687', count: 1, }, { val: '37.783368, -122.439666',
count: 1, }, ] }, "geo_loc_coord.raw": { "id": "geo_loc_coord.raw", "field":
"geo_loc_coord.raw", "filters": [{ "field": "geo_loc_coord.raw", "count": 39,
"facts": {}, "val": "51.4964, -0.1224", "rel": [] }, { "field":
"geo_loc_coord.raw", "count": 3, "facts": {}, "val": "52.2599, 0.2288", "rel": [] }
], "params": {}, "display": "geo_loc_coord.raw", "max-count": 39.0, "min-count":
0.0, "sum-of-counts": 42.0 }, }, results: [] } }]);
```

Result



esri:webmap

Description

Adds webmap layers from an esri webmap. The webmap id must be applied to the geo-map element.

Usage

as element:

```
<esri:webmap [url="{string}"] [toggle-layers="{boolean}"]> </esri:webmap>
```

Parameters

Param	Type	Details
url *(optional)*	**string**	The URL to the arcgis portal, if not supplied the default will be used by esri.
toggle-layers *(optional)*	**boolean**	Whether layers can be turned on and off.

Example

Source

```
<h3 id="example_esri-map">Esri Map</h3> <geo:map response="response2"
provider="esri" address-field="address" label-field="name" webmap-
id="ef9c7fbda731474d98647bebb4b33c20" zoom-level="13"> <esri:webmap toggle-
layers="true"></esri:webmap> </geo:map>
```

```
angular.module('lightning') .controller('ExampleController', ['$scope','$timeout',
function($scope,$timeout) { $scope.response2 = { page: 2, query: { rpp: 50 },
facets: {}, results: [ { result: { fields: { name:{ val:'Twigkit CA', }, address: {
val:'1551 McCarthy Boulevard, Suite 209, Milpitas, CA 95035', }, lat:{
val:'37.442172' }, long:{ val:'-121.9151737' } } }, }], } }]);
```

Result

Esri Map
World Reference Overlay
2517
layer0
Portland
405
30
26
esri

facet:dropdown

Description

Display a facet as a select dropdown, allowing the user to apply one or more filters by selection options from the dropdown. To specify the facet that should be rendered, either pass in a facet object directly or specify a facet-name and a response from where to retrieve that named facet.

Usage

as element:

```
<facet:dropdown facet="{string}" response="{string}" facet-name="{string}"
[select="{string}"] display-label="{string}" [search-enabled="{boolean}"]>
</facet:dropdown>
```

Parameters

Param	Type	Details
facet	**string**	A facet object to display
response	**string**	The name of the response to use.
facet-name	**string**	Facet name to use from the response object

select *(optional)*	**string**	How to render each facet filter
		`+`
		`Valid values:-`
		• **multiselect**
		Supports selecting multiple filters that get combined with an 'OR'. * **toggle**
		Only allows one filter to be selected at a time but shows unselected filters and counts * **multiselect-instant**
		Supports selecting multiple filters that get combined with an 'OR', filters will be applied automatically.

display-label	**string**	Initial display value for dropdown
search-enabled *(optional)*	**boolean**	Whether to enable search on the facet. If no query or platform is defined on the facet or facet-list it will enable the user to search through the filters that have been returned in the response. If a platform and query is defined on the facet or facet-list then it will search for filters in the search index. (Requires a tokenized field, if not using a tokenized field, if not using a tokenized field please see facet-field).

facet:hierarchical

Description

Hierarchical Facets for Solr.

In order to correctly display a taxonomy as a hierarchical facet you need to configure the requisite Twigkit service to retrieve the data structure [configured in your Solr index][1].

First you need to create a service configuration:

```
/conf
    /services
        /hierarchical
            /hierarchical.conf
```

The file 'hierarchical.conf' should be configured similar to the following example:

```
name: twigkit.search.solr.service.facet.GreedySolrHierarchicalFacetService
platform: platforms.solr
parents-suffix: _parent
```

platform This specifies the configured platform where the taxonomy has been indexed.

parents-suffix The suffix, that if appended to the facet's fieldname will designate the field containing parent node data. Will depend on how the taxonomy data has been indexed.

Example:

```
<search:platform var="platformHierarchical" conf="services.hierarchical">
</search:platform>
<search:facet facet-name="location" show="6" show-more="6">
    <facet:hierarchical facet-name="location" platform="platformHierarchical"
query="query"></facet:hierarchical>
</search:facet>
```

Usage

as element:

```
<facet:hierarchical query="{string}" [facet-name="{string}"] [platform="{string}"]
[count-number="{string}"] [select="{string}"] [max-characters="{Number}"] [expand-
root="{boolean}"]> </facet:hierarchical>
```

Directive info

- This directive creates new scope.

Parameters

Param	Type	Details
query	**string**	The name of the query object.
facet-name *(optional)*	**string**	Facet name to use from the response object
platform *(optional)*	**string**	The name of the platform to use.
count-number *(optional)*	**string**	Specify how the number representing the count should be formatted. Optional values are 'plain', 'formatted' and 'rounded'. Rounded means the number is represented in a short format (2k for ~2000). Formatted means that the number is comma formatted (2,000,000 for 2000000).Default: 'formatted'
select *(optional)*	**string**	How to render each FacetFilter. 'click': User selects a filter by clicking the value; and 'mutliselect': Supports selecting multiple filters that get combined with an 'OR'. Default: 'click'
max-characters *(optional)*	**Number**	Limit display value to a certain number of characters, adding '…' if maxCharacters is exceeded.
expand-root *(optional)*	**boolean**	If only one root filter exists, if this is set to true it will expand this filter showing the children. (Default:false).

Example

Source

```
<search:platform var="platform" conf="platforms.workflow.data.collection.google-directory"></search:platform> <search:query var="query" parameters='q' results-per-page="1"></search:query> <search:facet-list response="response" styling="facet-list facet-list-step facet-list-wrappedheader facet-list-wrappedheader-noborder">
<search:facet facet-name="test1" title="First Pass Files" collapsible="true">
<facet:hierarchical facet-name="test1" platform="platform" query="query">
</facet:hierarchical> </search:facet> </search:facet-list>
```

```
angular.module('lightning') .controller('ExampleController',
['$scope','$timeout','ResponseService', function($scope,$timeout,ResponseService) {
$scope.response = { page: 2, query: { rpp: 50 }, facets: { test: { filters: [ {
val: { dsp: 'Day 1', act: 'Day 1', }, count: 100, }, { val: { dsp: 'Day 2', act:
'Day 2', }, count: 200, } ] }, }, results: [] }
ResponseService.setResponse('response', $scope.response); }]);
```

geo:heatmap

Description

The heatmap directive

Usage

as element:

```
<geo:heatmap response="{string}" facet-name="{string}" [height="{string}"] [width="
{string}"] [scroll-to-zoom="{boolean}"] [options="{object}"]> </geo:heatmap>
```

Directive info

- This directive creates new scope.

Parameters

Param	Type	Details
response	**string**	Name of the response.
facet-name	**string**	Facet containing the data.
height *(optional)*	**string**	The height of the map container. Default is 400px.
width *(optional)*	**string**	The width of the map container. Default is 100%.
scroll-to-zoom *(optional)*	**boolean**	Whether the scroll wheel should control the map zoom. Disabled by default.
options *(optional)*	**object**	Extend the current options for the Google Map (Options Here) *(https://developers.google.com/maps/documentation/javascript/reference#MapOptions)*

geo:highmaps

Description

Highmaps directive for mapping data onto a map.

In order to use this directive you need to include the map you wish to use, they can be found here *(http://code.highcharts.com/mapdata/)*

For example if you want to use this map of the UK *(http://jsfiddle.net/gh/get/jquery/1.11.0/highslide-software/highcharts.com/tree/master/samples/mapdata/countries/gb/gb-all)* you need to include the following script tag `<script src="https://code.highcharts.com/mapdata/countries/gb/gb-all.js"></script>` and put `countries/gb/gb-all` on the chart highmaps map attribute. eg. `<chart:highmaps response="response" facet-name="facet" map="countries/gb/gb-all"></chart:highmaps>`

Usage

as element:

```
<geo:highmaps response="{string}" facet-name="{string}" map="{string}" [key="
{string}"] [title="{string}"] [subtitle="{string}"] [series-name="{string}"]
[query="{string}"] [options="{object}"] [select="{string}"]> </geo:highmaps>
```

Parameters

Param	Type	Details
response	**string**	Name of the response object to get the facet from.
facet-name	**string**	Name of the facet to retrieve the data from.
map	**string**	The map to use to render the data on.
key *(optional)*	**string**	The property for joining the data. (Default: 'hc-key') More details here *(http://www.highcharts.com/docs/maps/map-collection)*
title *(optional)*	**string**	Title for the chart.
subtitle *(optional)*	**string**	Subtitle for the chart.
series-name *(optional)*	**string**	Name of the series. (Default: Name from the facet)
query *(optional)*	**string**	Name of the query to apply filters to.
options *(optional)*	**object**	Extend the current options. Options can be found here. *(http://api.highcharts.com/highmaps/)*

select *(optional)*	**string**	How the data should be filtered when clicking on a part of the chart, a valid query must be present.
		`+`
		`Valid values:-`
		• **click**
		User selects a filter by clicking the value;
		(default) * **multiselect**
		Supports selecting multiple filters that get combined with an 'OR'. * **multiselect-instant**
		Supports selecting multiple filters that get combined with an 'OR', filters will be applied automatically.

geo:layer

Description

The geo:layer directive allows you to add multiple results from multiple responses to your *Geo Map*.

Usage

as element:

```
<geo:layer response="{string}" [address-field="{string}"] [latitude-field="
{string}"] [longitude-field="{string}"] [target="{string}"] [url-field="{string}"]
[url-prefix="{string}"] [label-field="{string}"] [icon="{string}"] [icon-field="
{string}"]> </geo:layer>
```

Directive info

- This directive creates new scope.

Parameters

Param	Type	Details
response	**string**	Name of the response.
address-field *(optional)*	**string**	The field containing a full address to geocode to obtain latitude and longitude values. Use this or latitudeField/longitudeField, not both.
latitude-field *(optional)*	**string**	The field containing the latitude.
longitude-field *(optional)*	**string**	The field containing the longitude.
target *(optional)*	**string**	Specify the link target to use for the map pins. Can be one of '_blank', '_parent', '_self', '_top' or the name of a window. Default value is '_blank'."
url-field *(optional)*	**string**	The field containing the URL to make clickable from the map marker.
url-prefix *(optional)*	**string**	Use this if the URL retrieved from the URLField needs to be prefixed with a value.
label-field *(optional)*	**string**	The field containing the label to use as tooltip for the marker.
icon *(optional)*	**string**	The location of the marker image to use for this set of coordinates.
icon-field *(optional)*	**string**	The field from the response to get the icon from.

geo:map

Description

The geoMap directive provides an abstraction layer to other map directives.

Google Maps

API Key

To add the API for Google Maps you need to include a constant on your index page like so with your API Key. `<helper:constant name="googleMapApi" value="YOUR_API_KEY_HERE"></helper:constant>` Doing this means you do not have to add the Google Map scripts to your index page

Esri Maps

For ESRI you will need to include the CSS and JavaScript from ESRI on your index page.

```
<script type="text/javascript" src="file:///Users/dustinguericke/repositories/fusion-
docs-site-legacy/public//js.arcgis.com/3.17/"></script>
<link rel="stylesheet" href="//js.arcgis.com/3.17/esri/css/esri.css">
```

Usage

as element:

```
<geo:map provider="{string}" [response="{string}"] [address-field="{string}"]
[latitude-field="{string}"] [longitude-field="{string}"] [api-key="{string}"]
[zoom-level="{string}"] [height="{string}"] [width="{string}"] [scroll-to-zoom="
{boolean}"] [target="{string}"] [url-field="{string}"] [url-prefix="{string}"]
[label-field="{string}"] [loading-overlay="{boolean}"] [icon="{string}"] [icon-
field="{string}"] [weight-field="{string}"] [geocode-server-url="{string}"]
[javascript-api-url="{string}"] [basemap="{string}"] [webmap-id="{string}"]
[options="{object}"] [events="{string}"]> </geo:map>
```

Directive info

- This directive creates new scope.

Parameters

Param	Type	Details
provider	**string**	Map provider to use (Supported values: google, esri)
response *(optional)*	**string**	Name of the response
address-field *(optional)*	**string**	The field containing a full address to geocode to obtain latitude and longitude values. Use this or latitudeField/longitudeField, not both.
latitude-field *(optional)*	**string**	The field containing the latitude.
longitude-field *(optional)*	**string**	The field containing the longitude.
apiKey *(optional)*	**string**	An API key to use with the map platform (if applicable)
zoom-level *(optional)*	**string**	The default zoom level to use for the map. Default is 13 for single results, with a group of results the map will scale automatically to fit the pins into the bounds, zoom level will be applied if the pins still fit within the new bounds..
height *(optional)*	**string**	The height of the map container. Default is 400px.
width *(optional)*	**string**	The width of the map container. Default is 100%.
scroll-to-zoom *(optional)*	**boolean**	Whether the scroll wheel should control the map zoom. Disabled by default.
target *(optional)*	**string**	Specify the link target to use for the map pins. Can be one of '_blank', '_parent', '_self', '_top' or the name of a window. Default value is '_blank'."
url-field *(optional)*	**string**	The field containing the URL to make clickable from the map marker.
url-prefix *(optional)*	**string**	Use this if the URL retrieved from the URLField needs to be prefixed with a value.

Param	Type	Details
label-field *(optional)*	**string**	The field containing the label to use as tooltip for the marker.
loading-overlay *(optional)*	**boolean**	Whether to show an overlay when the map is loading new points.
icon *(optional)*	**string**	The location of the marker image to use for this set of coordinates.
icon-field *(optional)*	**string**	The field from the response to get the icon from.
weight-field *(optional)*	**string**	The field containing information about the weight of the result. (Google Only)
geocode-server-url *(optional)*	**string**	The URL of the geocode server to turn addresses into coordinates (Esri only)
javascript-api-url *(optional)*	**string**	The URL with the location of the JavaScript API (Esri only must be 3.15 or greater)
basemap *(optional)*	**string**	The map to use to display the points on (Esri and Google only) Google defaults to: ROADMAP Google Options - HYBRID, ROADMAP, SATELLITE, TERRAIN. Esri defaults to: topo - Esri options *(https:///developers.arcgis.com/javascript/latest/api-reference/)*
webmap-id *(optional)*	**string**	The id of the web map to add to the esri map. (Only for esri, you must include the *esri webmap tag* within the geo:map directive).
options *(optional)*	**object**	Extend the current options for the Map (Google Only) (Options Here) *(https://developers.google.com/maps/documentation/javascript/reference#MapOptions)*

Param	Type	Details
events *(optional)*	**string**	Adds support for filtering results based on the current map position (apply), on zoom (zoom), on drag (drag) and also when drawing a rectangle (draw). (You must define latitude and longitude fields for this to work). Valid values: • **draw** Allows the user to draw a box on the map which will then return results within the box. • **apply** Allows the user to search the currently visible area. (Google only) • **zoom** When the user zooms in or out of an area the event "zoom_changed_gmaps" will fire with an object containing the coordinates of the box. (Google only) • **drag** When the user drags the map and the selected area changes the event "dragend_gmaps" will fire with an object containing the coordinates of the box. (Google only)

graph:flow

Description

The Graph Flow Tag displays network data rendered as a Sankey diagram. The body of the Graph Flow tag can contain a *Chart Credit Tag* for adding a credit to the chart and the *Chart Export Tag* for exporting the chart.

For detail on how to change the color and background color see the demo below, for more detail on what you can change and set using CSS see here. *(http://www.highcharts.com/docs/chart-design-and-style/style-by-css)*

The following tags can be nested within the chart sankey tag.

- *Chart Credit Tag*
- *Chart Export Tag*

Usage

as element:

```
<graph:flow platform="{string}" query="{string}" source-field="{string}" target-field="{string}" [measure="{string}"] [measure-field="{string}"] data="{object}" [title="{string}"] [subtitle="{string}"] [series-name="{string}"] [inverted="{boolean}"] [options="{object}"]> </graph:flow>
```

Directive info

- This directive creates new scope.

Parameters

Param	Type	Details
platform	**string**	Name of the platform (Required if not using `data` attribute).
query	**string**	Name of the query (Required if not using `data` attribute).
source-field	**string**	Name of a primary field to facet on (for example "Country"). The values of this facet will define the set of graph nodes with outgoing edges. (Required if not using `data` attribute).
target-field	**string**	Name of a secondary field to facet on (for example "City"), for every facet value of the primary field. The values of this facet will define the set of graph nodes with incoming edges. For example, say the primary facet (source-field) has values for Country=Germany and Country=France and the secondary facet (target-field) has values for City=Berlin, City=Munich and City=Paris. Then we add graph edges Germany-→Berlin, Germany-→Munich and France-→Paris. (Required if not using `data` attribute).
measure *(optional)*	**string**	Name of a numerical measure that defines the weight of each graph edge. When omitted, this defaults to the counts of the facets defined by the target-field. Some possible measures include "sum", "mean" and "min". (Only applicable if not using `data` attribute).
measure-field *(optional)*	**string**	Field to measure, when the "measure" option is specified.

data	**object**	Array of arrays containing the data to render the sankey chart. In the format of `['from', 'to', 'weight']`
		`[`
		`  ['Spain', 'Portugal', 100]`
		`  ['Spain', 'Brazil', 300]`
		`]`
		(Required if not using the `platform` attribute)

title *(optional)*	**string**	Chart title

Param	Type	Details
subtitle *(optional)*	**string**	Chart sub title
series-name *(optional)*	**string**	The name of the series. *(default: '')*
inverted *(optional)*	**boolean**	Whether the chart should be inverted. *(default: true)*
options *(optional)*	**object**	Extend the current options. Options can be found here. *(http://api.highcharts.com/highcharts/)*

helper:constant

Description

The constant tagf adds a constant to the application.

Usage

as element:

```
<helper:constant name="{string}" value="{string}"> </helper:constant>
```

Directive info

- This directive creates new scope.

Parameters

Param	Type	Details
name	**string**	The name of the constant
value	**string**	The value of the constant

Example

Source

```
<helper:constant name="test" value="banana"></helper:constant> <span>My Constant:
{{test}}</span>
```

```
angular.module('lightning') .controller('ExampleController', ['$scope','$twigkit',
function($scope,$twigkit) { $scope.test = $twigkit.getConstant('test','noValue');
}]);
```

Result

My Constant: banana

helper:set

Description

The Set directive sets a variable in the view to a new variable in the view controller.

Usage

as element:

```
<helper:set var="{string}" value="{string}"> </helper:set>
```

Directive info

- This directive creates new scope.

Parameters

Param	Type	Details
var	**string**	The name of the new variable
value	**string**	The variable to copy

Example

Source

```
<helper:set var="myText" value="'My Text Result'"></helper:set> <p><strong>myText:
</strong> {{myText}}</p>
```

Result

myText: My Text Result

helper:title

Description

The title directive allows you to set the title for the page.

Usage

as element:

```
<helper:title title="{string}"> </helper:title>
```

Directive info

- This directive creates new scope.

Parameters

Param	Type	Details
title	**string**	The title of the page.

Example

```
<helper:title title="Twigkit Application" </search:title>
```

hide:when

Description

Show or hideWhen element based on expression passed to it.

Usage

as attribute

```
<ANY hide-when="{expression}"> ... </ANY>
```

Parameters

Param	Type	Details
hide-when	**expression**	Expression to evaluate.

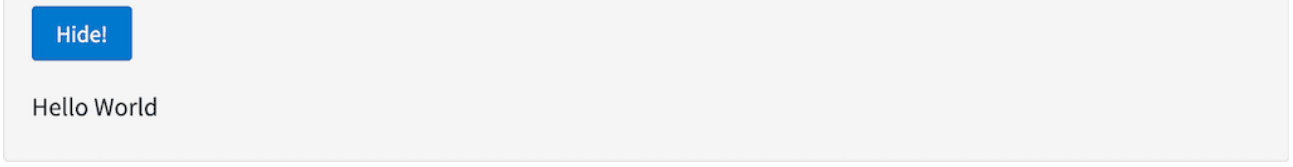
Example

Source

```
<p><a href="javascript:void(0);" ng-click="alert()" styling="button">{{foo ? 'Show' : 'Hide'}}!</a></p> <h3 hide-when="foo" id="example_hello-world">Hello World</h3>
```

```
angular.module('lightning') .controller('ExampleController', ['$scope','$timeout', function($scope,$timeout) { $scope.foo = false; $scope.alert = function(){ $scope.foo = !$scope.foo }; }]);
```

Result



instant:result-list

Description

The Instant Result List directive allows you to show results matching the users typed query under the search box.

Caveats

- You cannot use an `ng-if` on a direct child of the `<instant:result-list>` tag, instead wrap your `ng-if` with a `<div>` tag. Example:-

Do Not!

```
<search:box>
    <instant:result-list>
        <search:result ng-if="type == 'foo'"></search:result>
        <search:result ng-if="type == 'bar'"></search:result>
    </instant:result-list>
</search:box>
```

Do

```
<search:box>
    <instant:result-list>
        <div>
            <search:result ng-if="type == 'foo'"></search:result>
            <search:result ng-if="type == 'bar'"></search:result>
        </div>
    </instant:result-list>
</search:box>
```

Usage

as element:

```
<instant:result-list [platform="{string}"] [query="{string}"] [children-selector="
{string}"] [wildcard="{string}"]> </instant:result-list>
```

Directive info

- This directive creates new scope.

Parameters

Param	Type	Details
platform *(optional)*	**string**	The platform to get the results from, if no platform is defined it will try and use a platform called 'platform'.
query *(optional)*	**string**	The query to use against the platform. If the search-within checkbox is checked the filters of this query will be reset. Also if rpp is set on the query then this will be used when getting back the results, otherwise it will be set to 5.
children-selector *(optional)*	**string**	A selector for selecting the children elements that should be used for the arrow key selection mechanism. Should be used when the children to be selected are not the immediate children of resultList.
wildcard *(optional)*	**string**	Whether to add a wildcard to the search query. Valid values: • **left** - `*searchQuery` • **right** ***(default)*** - `searchQuery*` • **both** - `*searchQuery*` • **none** - `searchQuery`

layout:animate

Description

Creates an animation when the value (on) is available.

Usage

as element:

```
<layout:animate animation="{string}" on="{object}"> </layout:animate>
```

as attribute

```
<ANY layout-animate animation="{string}" on="{object}"> ... </ANY>
```

Parameters

Param	Type	Details
animation	**string**	The animation to use when the value is available
on	**object**	The object to watch for a value

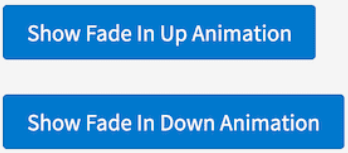
Example

Source

```
<button class="tk-stl-button" ng-click="fadeInUp()">Show Fade In Up
Animation</button> <layout:animate animation="animate-fadeInUp" on="fadeInUpValue">
<p>Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor
incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud
exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute
irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla
pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia
deserunt mollit anim id est laborum.</p> </layout:animate> <br /><br /> <button
class="tk-stl-button" ng-click="fadeInDown()">Show Fade In Down Animation</button>
<layout:animate animation="animate-fadeInDown" on="fadeInDownValue"> <p>Lorem ipsum
dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut
labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation
ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in
reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur.
Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt
mollit anim id est laborum.</p> </layout:animate>
```

```
angular.module('lightning') .controller('ExampleController', ['$scope',
function($scope) { $scope.fadeInUp = undefined; $scope.fadeInUp = function(){
if($scope.fadeInUpValue == undefined){ $scope.fadeInUpValue = 'A Value'; } else {
$scope.fadeInUpValue = undefined; } } $scope.fadeInDown = function(){
if($scope.fadeInDownValue == undefined){ $scope.fadeInDownValue = 'A Value'; } else
{ $scope.fadeInDownValue = undefined; } } }]);
```

Result



layout:block

Description

Adds grid block classes to element

Usage

as element:

```
<layout:block [xs="{size}"] [sm="{size}"] [md="{size}"] [lg="{size}"] [xl="{size}"] [xs-offset="{size}"] [sm-offset="{size}"] [md-offset="{size}"] [lg-offset="{size}"] [xl-offset="{size}"] [dynamic="{string}"]> </layout:block>
```

as attribute

```
<ANY layout-block [xs="{size}"] [sm="{size}"] [md="{size}"] [lg="{size}"] [xl="{size}"] [xs-offset="{size}"] [sm-offset="{size}"] [md-offset="{size}"] [lg-offset="{size}"] [xl-offset="{size}"] [dynamic="{string}"]> ... </ANY>
```

Parameters

Param	Type	Details
xs *(optional)*	**size**	The grid ratio to use for the smallest break point
sm *(optional)*	**size**	The grid ratio to use for the small break point
md *(optional)*	**size**	The grid ratio to use for the medium break point
lg *(optional)*	**size**	The grid ratio to use for the large break point
xl *(optional)*	**size**	The grid ratio to use for above the large break point
xs-offset *(optional)*	**size**	The offset ratio to use for the smallest break point
sm-offset *(optional)*	**size**	The offset ratio to use for the small break point
md-offset *(optional)*	**size**	The offset ratio to use for the medium break point
lg-offset *(optional)*	**size**	The offset ratio to use for the large break point
xl-offset *(optional)*	**size**	The offset ratio to use for above the large break point
dynamic *(optional)*	**string**	Watch for updates to the attributes

Example

Source

```
<layout:grid> <layout:block sm="23-24" md="3-4" lg="1-2" xl="1-4" lg-offset="1-4" xl-offset="1-2">Content</layout:block> </layout:grid>
```

layout:box

Description

Creates a box

Usage

as element:

```
<layout:box> </layout:box>
```

layout:center-content

Description

Creates a centered content area with some padding

Usage

as element:

```
<layout:center-content> </layout:center-content>
```

layout:content

Description

Creates a content area

Usage

as element:

```
<layout:content> </layout:content>
```

layout:controls

Description

The `<layout:controls>` tag wraps control elements into a group making all the controls on one line or individually for a control to be on a separate line.

Layout controls is required for the following directives:-

- *Sort Tag*
- *Results Per Page Tag*
- *Fields Tag*

Usage

as element:

```
<layout:controls> </layout:controls>
```

layout:grid

Description

Adds grid classes to element

Usage

as element:

```
<layout:grid [fixed="{string}"]> </layout:grid>
```

as attribute

```
<ANY layout-grid [fixed="{string}"]> ... </ANY>
```

Parameters

Param	Type	Details
fixed *(optional)*	**string**	If an attribute of 'fixed' is on the element a fixed grid will be uses instead

layout:include

Description

The include directives allows you to include a template on the page.

Usage

as element:

```
<layout:include file="{string}"> </layout:include>
```

Directive info

- This directive creates new scope.

Parameters

Param	Type	Details
file	**string**	The location of the file of the template

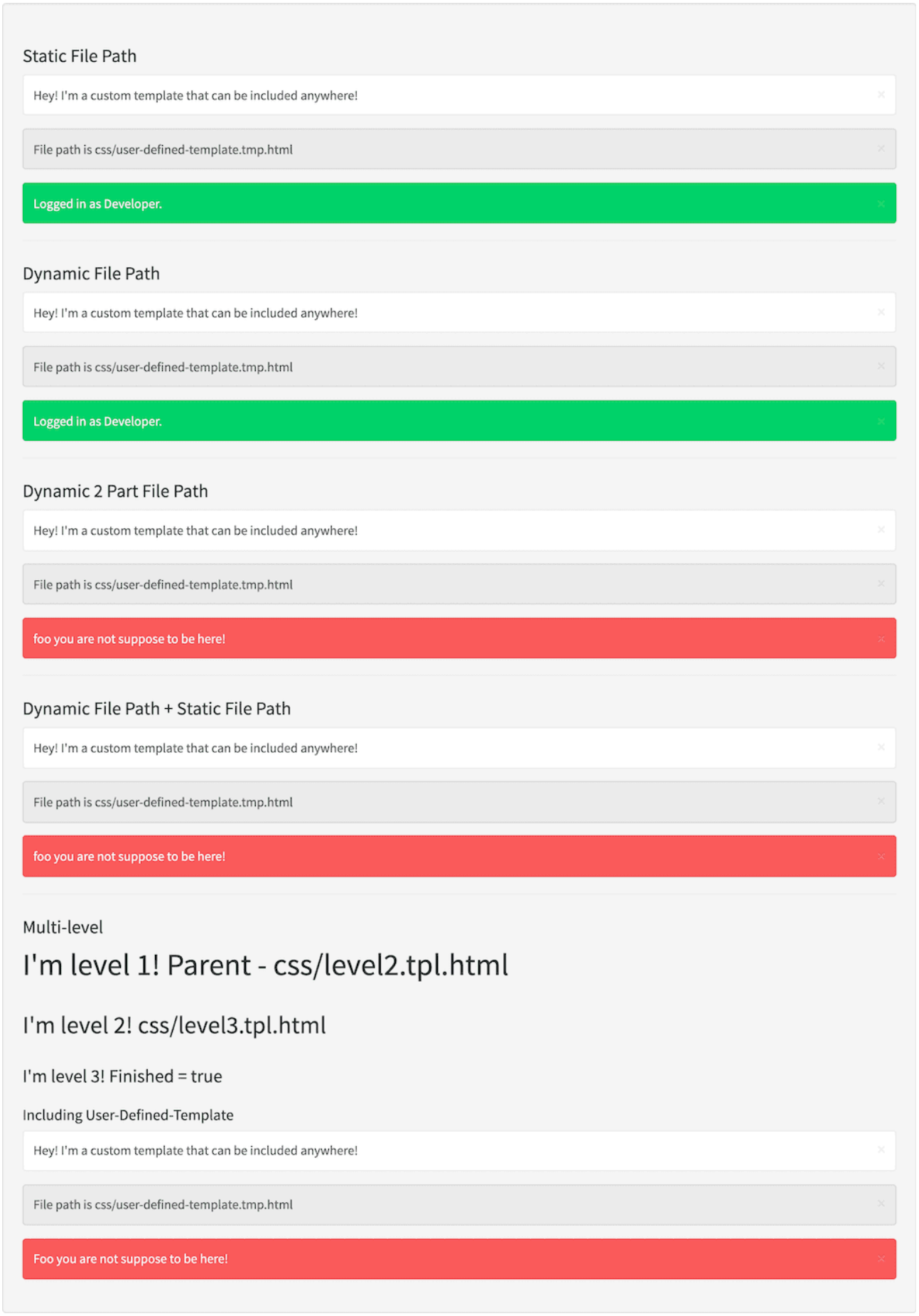
Example

Source

```
<h3 id="example_static-file-path">Static File Path</h3> <layout:include
file="css/user-defined-template.tmp.html" user="{{user}}" file-path="{{filePath}}">
</layout:include> <hr /> <h3 id="example_dynamic-file-path">Dynamic File Path</h3>
<layout:include file="{{filePath}}" user="{{user}}" file-path="{{filePath}}">
</layout:include> <hr /> <h3 id="example_dynamic-2-part-file-path">Dynamic 2 Part
File Path</h3> <layout:include file="{{partFilePath1 + partFilePath2}}" user="foo"
file-path="{{filePath}}"></layout:include> <hr /> <h3 id="example_dynamic-file-
path-+-static-file-path">Dynamic File Path + Static File Path</h3> <layout:include
file="{{partFilePath1 + 'tmp.html'}}" user="foo" file-path="{{filePath}}">
</layout:include> <hr /> <h3 id="example_multi-level">Multi-level</h3>
<layout:include file="css/level1.tpl.html" leveltype="Parent"
child="css/level2.tpl.html"></layout:include>
```

```
angular.module('lightning') .controller('ExampleController', ['$scope',
function($scope) { $scope.user = 'Developer'; $scope.filePath = 'css/user-defined-
template.tmp.html'; $scope.partFilePath1 = 'css/user-defined-template.';
$scope.partFilePath2 = 'tmp.html'; }]);
```

Result


Static File Path
Hey! I'm a custom template that can be included anywhere!
File path is css/user-defined-template.tmp.html
Logged in as Developer.
Dynamic File Path
Hey! I'm a custom template that can be included anywhere!
File path is css/user-defined-template.tmp.html
Logged in as Developer.
Dynamic 2 Part File Path
Hey! I'm a custom template that can be included anywhere!
File path is css/user-defined-template.tmp.html
foo you are not suppose to be here!
Dynamic File Path + Static File Path
Hey! I'm a custom template that can be included anywhere!
File path is css/user-defined-template.tmp.html
foo you are not suppose to be here!
Multi-level
I'm level 1! Parent - css/level2.tpl.html
I'm level 2! css/level3.tpl.html
I'm level 3! Finished = true
Including User-Defined-Template
Hey! I'm a custom template that can be included anywhere!
File path is css/user-defined-template.tmp.html
Foo you are not suppose to be here!

layout:logo

Description

Creates a content area

Usage

as element:

```
<layout:logo [image="{string}"] [image-link="{string}"] [height="{string}"]
[width="{string}"] [alt="{string}"]> </layout:logo>
```

Parameters

Param	Type	Details
image *(optional)*	**string**	The URL of the image to display.
image-link *(optional)*	**string**	The URL to redirect to when the image is click. Default: '/'.
height *(optional)*	**string**	The height of the logo in px.
width *(optional)*	**string**	The width of the logo in px.
alt *(optional)*	**string**	The alternative text if the image cannot be displayed.

Example

Source

```
<layout:logo image-link="https://twigkit.com"
image="//static1.squarespace.com/static/53da228ce4b052fe60de3a3c/t/548af226e4b0dd3c
b0252f2b/1466005538473/?format=1500w" height="34px" alt="Logo"></layout:logo>
```

Result

layout:sidebar

Description

Creates a sidebar area

Usage

as element:

```
<layout:sidebar> </layout:sidebar>
```

lifecycle:delay

Description

Delays rendering of an object

Usage

as element:

```
<lifecycle:delay delay="{string}"> </lifecycle:delay>
```

Parameters

Param	Type	Details
delay	**string**	The delay in ms

Example

Source

```
<p>A message will appear after 5 seconds has passed.</p> <lifecycle:delay
delay="5000"> <search:alert style="alertbox-positive">Alert - 5 Seconds has passed.
</search:alert> </lifecycle:delay> <p>A message will appear after 10 seconds has
passed.</p> <lifecycle:delay delay="10000"> <search:alert style="alertbox-
warning">Alert - 10 Seconds has passed.</search:alert> </lifecycle:delay> <p>A
message will appear after 20 seconds has passed.</p> <lifecycle:delay
delay="20000"> <search:alert style="alertbox-negative">Alert - 20 Seconds has
passed.</search:alert> </lifecycle:delay>
```

Result

A message will appear after 5 seconds has passed.

A message will appear after 10 seconds has passed.

A message will appear after 20 seconds has passed.

lifecycle:display

Description

Display an element for a period of time

Usage

as element:

```
<lifecycle:display time="{string}"> </lifecycle:display>
```

Parameters

Param	Type	Details
time	**string**	The time in ms to display the elements

Example

Source

```
<p>A message will disappear after 5 seconds.</p> <lifecycle:display time="5000">
<search:alert style="alertbox-negative">Alert - I disappear after 5 seconds.
</search:alert> </lifecycle:display> <p>A message will disappear after 10 seconds
has passed.</p> <lifecycle:display time="10000"> <search:alert style="alertbox-
warning">Alert - I disappear after 10 seconds.</search:alert> </lifecycle:display>
<p>A message will disappear after 20 seconds has passed.</p> <lifecycle:display
time="20000"> <search:alert style="alertbox-positive">Alert - I disappear after 20
seconds.</search:alert> </lifecycle:display>
```

Result

lifecycle:event

Description

Broadcasts an event

Usage

as element:

```
<lifecycle:event event="{string}" global="{boolean}"> </lifecycle:event>
```

Parameters

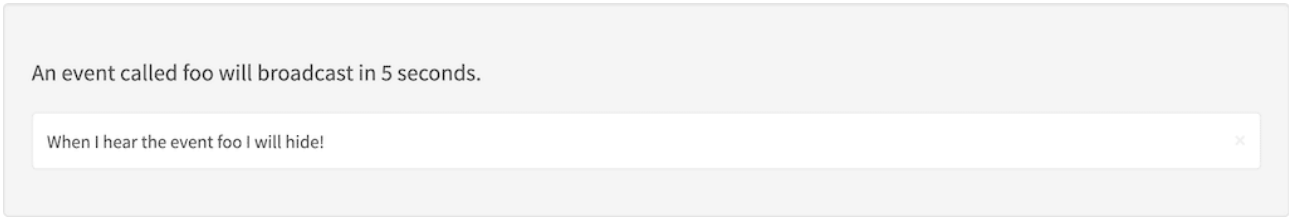
Param	Type	Details
event	**string**	The name of the event to broadcast
global	**boolean**	If true the event is broadcasted to the entire application, otherwise it will just broadcast to it's children. (Default: true)

Example

Source

```
<p>An event called foo will broadcast in 5 seconds.</p> <lifecycle:delay
delay="5000"> <lifecycle:event event="foo"> <search:alert style="alertbox-
positive">Event foo has been broadcast</search:alert> </lifecycle:event>
</lifecycle:delay> <lifecycle:on event="foo"> <lifecycle:delay delay="1000">
<search:alert style="alertbox-positive">Event foo has been heard so I have
appeared!</search:alert> </lifecycle:delay> </lifecycle:on> <lifecycle:on
event="foo" hide="true"> <search:alert style="alertbox-warning">When I hear the
event foo I will hide!</search:alert> </lifecycle:on>
```

Result



lifecycle:on

Description

Shows or hides elements when an event is broadcast

Usage

as element:

```
<lifecycle:on event="{string}" hide="{boolean}" show="{boolean}"> </lifecycle:on>
```

Parameters

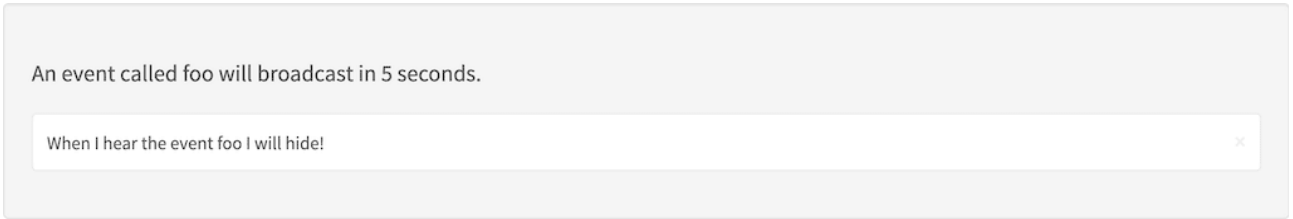
Param	Type	Details
event	**string**	The name of the event to be listened to
hide	**boolean**	If true when the event is heard the elements will be hidden.
show	**boolean**	If true when the event is heard the elements will be shown.

Example

Source

```
<p>An event called foo will broadcast in 5 seconds.</p> <lifecycle:delay
delay="5000"> <lifecycle:event event="foo"> <search:alert style="alertbox-
positive">Event foo has been broadcast</search:alert> </lifecycle:event>
</lifecycle:delay> <lifecycle:on event="foo"> <lifecycle:delay delay="1000">
<search:alert style="alertbox-positive">Event foo has been heard so I have
appeared!</search:alert> </lifecycle:delay> </lifecycle:on> <lifecycle:on
event="foo" hide="true"> <search:alert style="alertbox-warning">When I hear the
event foo I will hide!</search:alert> </lifecycle:on>
```

Result



lifecycle:required

Description

Renders objects when other object is available;

Usage

as element:

```
<lifecycle:required object="{string}"> </lifecycle:required>
```

Parameters

Param	Type	Details
object	**string**	The object in question

Example

Source

```
<button class="tk-stl-button" ng-click="toggle()">Toggle</button>
<lifecycle:required object="value"> <p>I only appear when "value" has a value.</p>
</lifecycle:required>
```

```
angular.module('lightning') .controller('ExampleController', ['$scope',
function($scope) { $scope.toggle = function(){ if($scope.value == undefined){
$scope.value = 'A Value'; } else { $scope.value = undefined; } } }]);
```

Result

media:csv

Description

The media:csv tag allows you to export results into csv format.

Usage

as element:

```
<media:csv conf="{string}" query="{object}" [fetch="{number|expression}"] [batch-size="{number}"] [delimiter="{string}"] [fields="{string}"]> </media:csv>
```

Directive info

- This directive creates new scope.

Parameters

Param	Type	Details
conf	**string**	The platform configuration to use e.g. platforms.gsa.foo
query	**object**	The query to submit to the CSV service
fetch *(optional)*	**number]link:[expression**	The number of results or rows to fetch.
batch-size *(optional)*	**number**	The number of results or rows to fetch in each batch.
delimiter *(optional)*	**string**	Delimiter to use in the CSV
fields *(optional)*	**string**	Fields to output, comma-separated or * for all.

media:excel

Description

The media:excel tag allows you to export results into excel format.

Usage

as element:

```
<media:excel conf="{string}" query="{object}" [fetch="{number|expression}"]
[fields="{string}"] [header="{string}"]> </media:excel>
```

Directive info

- This directive creates new scope.

Parameters

Param	Type	Details
conf	**string**	The platform configuration to use e.g. platforms.gsa.foo
query	**object**	The query to submit to the CSV service
fetch *(optional)*	**number]link: [expression**	The number of results or rows to fetch.
fields *(optional)*	**string**	Fields to output, comma-separated or * for all.
header *(optional)*	**string**	Replace field names in the header with these header values. Must be a comma separated list of equal length to the fields parameter

media:image

Description

The Image tag displays images from a given field in a result or a `src`. It will automatically resize and cache the image for performance.

The following module will need to be added to your pom.xml in order to use the twigkit image-service.

```
<dependency>
     <groupId>twigkit</groupId>
     <artifactId>twigkit.media</artifactId>
     <version>${project.parent.version}</version>
</dependency>
```

If your image is within an expression you will need to use the trusted filter eg (`<media:image src="{{imageUrl | trusted}}"></media:image>`)

If you are combining plain text with a variable it is recommend that you do this within a function in a controller. You can see this in the example below. This will remove the chance of getting a 404 in the JavaScript console caused by a request for the image being sent before the values in the src attribute were evaluated.

Example

Controller

```
$scope.myImageUrl = function(result){
   return 'http://placehold.it/' + result.fields.x.val + 'x' + result.fields.y.val;
}
```

View

```
<media:image src="{{myImageUrl(result) | trusted}}"></media:image
```

Usage

as element:

```
<media:image [result="{object}"] [field-name="{string}"] [field-prefix="{string}"] [field-suffix="{string}"] [src="{string}"] [lazy-load="{boolean}"] [fallback-image-url="{string}"] [width="{number}"] [height="{number}"] [alt="{string}"] [url="{string}"] [cache-original="{boolean}"] [image-service-url="{string}"] [use-headers="{boolean}"]> </media:image>
```

Parameters

Param	Type	Details
result *(optional)*	**object**	The result object the field with the image is coming from.
field-name *(optional)*	**string**	The field in the result to obtain the image URL.
field-prefix *(optional)*	**string**	If the field contains a relative URL, then prefix it with this value.
field-suffix *(optional)*	**string**	If the field contains a relative URL, then append this to that value.
src *(optional)*	**string**	Rather than specify a field-name, provide a URL to the desired image. If you are using an angular expression you will need to add the #/api/lightning.filter:trusted[trusted filter]. For example, (`<media:image src="{{imageUrl | trusted}}"></media:image>`)
lazy-load *(optional)*	**boolean**	Whether the images should be lazy loaded. Lazy loading works best if image size is defined. Default:false.
fallback-image-url *(optional)*	**string**	If no image is found, use the fallback URL provided. If no fallback image URL is provided and the image is not found no image will be displayed.
width *(optional)*	**number**	Width of the image to resize to. You can specify either width, height or both.
height *(optional)*	**number**	Height of the image to resize to. You can specify either width, height or both.
alt *(optional)*	**string**	Provide alternative text for the image.

Param	Type	Details
url *(optional)*	**string**	Provide a URL to make the image clickable.
cache-original *(optional)*	**boolean**	Whether the write the original image (prior to resizing) to cache. Defaults to true.
image-service-url *(optional)*	**string**	Specify an alternative URL for the Twigkit Media Module image service.
use-headers *(optional)*	**boolean**	Whether to pass HTTP headers to the underlying image service (default: false).

ng:compile

Description

Compiles HTML string. Works like ngBindHtml but compiles the string.

Usage

as attribute

```
<ANY ng-compile> ... </ANY>
```

Directive info

- This directive creates new scope.

ng:inject

Description

Transcludes the content and sets the scope of the transcluded content to that of the parent directive. ngTransclude would set the scope of the transcluded content to the scope of the view. More information on the reason for this directive can be found here: https://github.com/angular/angular.js/issues/7874#issuecomment-47644966
(https://github.com/angular/angular.js/issues/7874#issuecomment-47644966)

Usage

as element:

```
<ng:inject> </ng:inject>
```

as attribute

```
<ANY ng-inject> ... </ANY>
```

as class

```
<ANY class="ng-inject"> ... </ANY>
```

Directive info

- This directive creates new scope.

padding:bottom

Description

Adds a class for padding bottom based on attribute value

Usage

as attribute

```
<ANY padding-bottom="{number}"> ... </ANY>
```

Parameters

Param	Type	Details
padding-bottom	**number**	Amount of padding to include, acceptable values (0, 0.25, 0.5, 1 and 2)

Example

Source

```
<p style="bg-tertiary text-tertiary-contrast" padding-bottom="2"> Check out my padding at the bottom (padding-bottom="2") </p> <p style="bg-tertiary text-tertiary-contrast" padding-bottom="1"> Check out my padding at the bottom (padding-bottom="1") </p> <p style="bg-tertiary text-tertiary-contrast" padding-bottom="0.5"> Check out my padding at the bottom (padding-bottom="0.5") </p> <p style="bg-tertiary text-tertiary-contrast" padding-bottom="0.25"> Check out my padding at the bottom (padding-bottom="0.25") </p> <p style="bg-tertiary text-tertiary-contrast" padding-bottom="0"> Check out my padding at the bottom (padding-bottom="0") </p>
```

Result

Check out my padding at the bottom (padding-bottom="2")

Check out my padding at the bottom (padding-bottom="1")

Check out my padding at the bottom (padding-bottom="0.5")

Check out my padding at the bottom (padding-bottom="0.25")

Check out my padding at the bottom (padding-bottom="0")

padding:left

Description

Adds a class for padding left based on attribute value

Usage

as attribute

```
<ANY padding-left="{number}"> ... </ANY>
```

Parameters

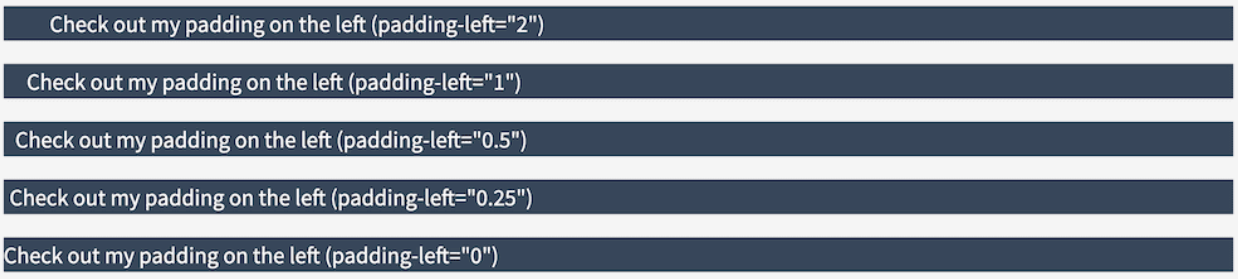
Param	Type	Details
padding-left	**number**	Amount of padding to include, acceptable values (0, 0.25, 0.5, 1 and 2)

Example

Source

```
<p style="bg-tertiary text-tertiary-contrast" padding-left="2"> Check out my padding on the left (padding-left="2") </p> <p style="bg-tertiary text-tertiary-contrast" padding-left="1"> Check out my padding on the left (padding-left="1") </p> <p style="bg-tertiary text-tertiary-contrast" padding-left="0.5"> Check out my padding on the left (padding-left="0.5") </p> <p style="bg-tertiary text-tertiary-contrast" padding-left="0.25"> Check out my padding on the left (padding-left="0.25") </p> <p style="bg-tertiary text-tertiary-contrast" padding-left="0"> Check out my padding on the left (padding-left="0") </p>
```

Result



padding:right

Description

Adds a class for padding right based on attribute value

Usage

as attribute

```
<ANY padding-right="{number}"> ... </ANY>
```

Parameters

Param	Type	Details
padding-right	**number**	Amount of padding to include, acceptable values (0, 0.25, 0.5, 1 and 2)

Example

Source

```
<p style="text-align:right;" class="tk-stl-bg-tertiary tk-stl-text-tertiary-contrast" padding-right="2"> Check out my padding on the right (padding-right="2") </p> <p style="text-align:right;" class="tk-stl-bg-tertiary tk-stl-text-tertiary-contrast" padding-right="1"> Check out my padding on the right (padding-right="1") </p> <p style="text-align:right;" class="tk-stl-bg-tertiary tk-stl-text-tertiary-contrast" padding-right="0.5"> Check out my padding on the right (padding-right="0.5") </p> <p style="text-align:right;" class="tk-stl-bg-tertiary tk-stl-text-tertiary-contrast" padding-right="0.25"> Check out my padding on the right (padding-right="0.25") </p> <p style="text-align:right;" class="tk-stl-bg-tertiary tk-stl-text-tertiary-contrast" padding-right="0"> Check out my padding on the right (padding-right="0") </p>
```

Result

padding:top

Description

Adds a class for padding top based on attribute value

Usage

as attribute

```
<ANY padding-top="{number}"> ... </ANY>
```

Parameters

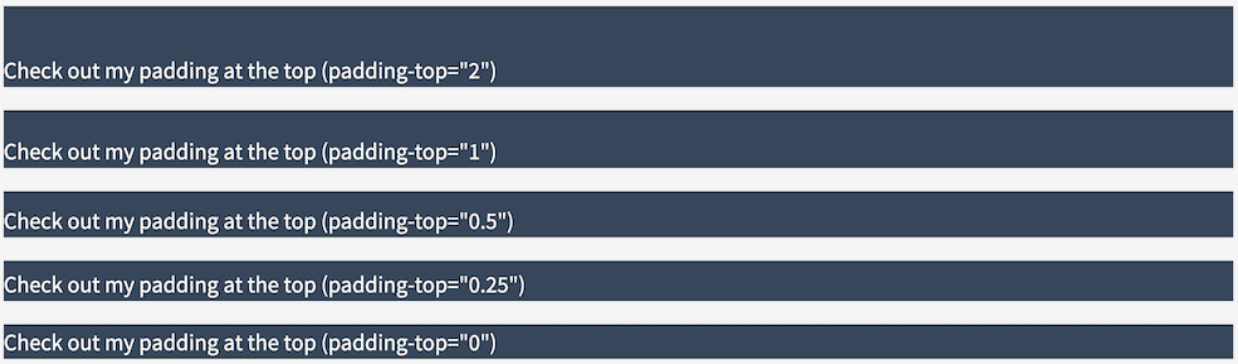
Param	Type	Details
padding-top	**number**	Amount of padding to include, acceptable values (0, 0.25, 0.5, 1 and 2)

Example

Source

```
<p style="bg-tertiary text-tertiary-contrast" padding-top="2"> Check out my padding at the top (padding-top="2") </p> <p style="bg-tertiary text-tertiary-contrast" padding-top="1"> Check out my padding at the top (padding-top="1") </p> <p style="bg-tertiary text-tertiary-contrast" padding-top="0.5"> Check out my padding at the top (padding-top="0.5") </p> <p style="bg-tertiary text-tertiary-contrast" padding-top="0.25"> Check out my padding at the top (padding-top="0.25") </p> <p style="bg-tertiary text-tertiary-contrast" padding-top="0"> Check out my padding at the top (padding-top="0") </p>
```

Result



pivot:table

Description

Display a facet with related filters as a table

Usage

as element:

```
<pivot:table platform="{string}" query="{string}" facet="{string}" expand-query="
{string}" expand-facet="{string}" [expand="{boolean}"] [field="{string}"]>
</pivot:table>
```

Parameters

Param	Type	Details
platform	**string**	The platform to query against
query	**string**	The query to use
facet	**string**	The facet to display
expand-query	**string**	The query to use for expanding
expand-facet	**string**	The facet to pivot on when expanding
expand *(optional)*	**boolean**	Allow expanding. Default: true
field *(optional)*	**string**	The filter field name to add to the query

progress:bar

Description

Creates a horizontal progress bar.

Usage

as element:

```
<progress:bar percentage="{string}"> </progress:bar>
```

Parameters

Param	Type	Details
percentage	**string**	The percentage progress

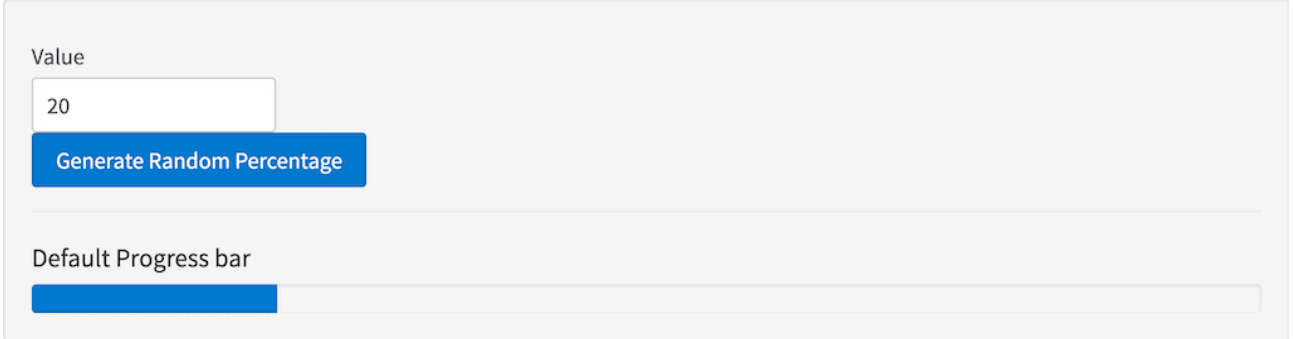
Example

Source

```
<label for="value">Value</label> <input id="value" type="number"
placeholder="Value" ng-model="value"><br /> <button ng-click="generateNewNumber()"
style="button">Generate Random Percentage</button> <hr /> <h3 id="example_default-
progress-bar">Default Progress bar</h3> <progress:bar percentage="value">
</progress:bar>
```

```
angular.module('lightning') .controller('ExampleController', ['$scope','$timeout',
function($scope,$timeout) { $scope.value = 20; $scope.generateNewNumber =
function(){ $scope.value = Math.floor(Math.random() * (100 - 0 + 1)); }; }]);
```

Result



progress:radial

Description

Creates a radial progress bar

Usage

as element:

```
<progress:radial percentage="{string}" [fill-class="{string}"] [percentage-class="{string}"]> </progress:radial>
```

Parameters

Param	Type	Details
percentage	**string**	The percentage progress
fill-class *(optional)*	**string**	Add a class to the fill elements
percentage-class *(optional)*	**string**	Add a class to the percentage element

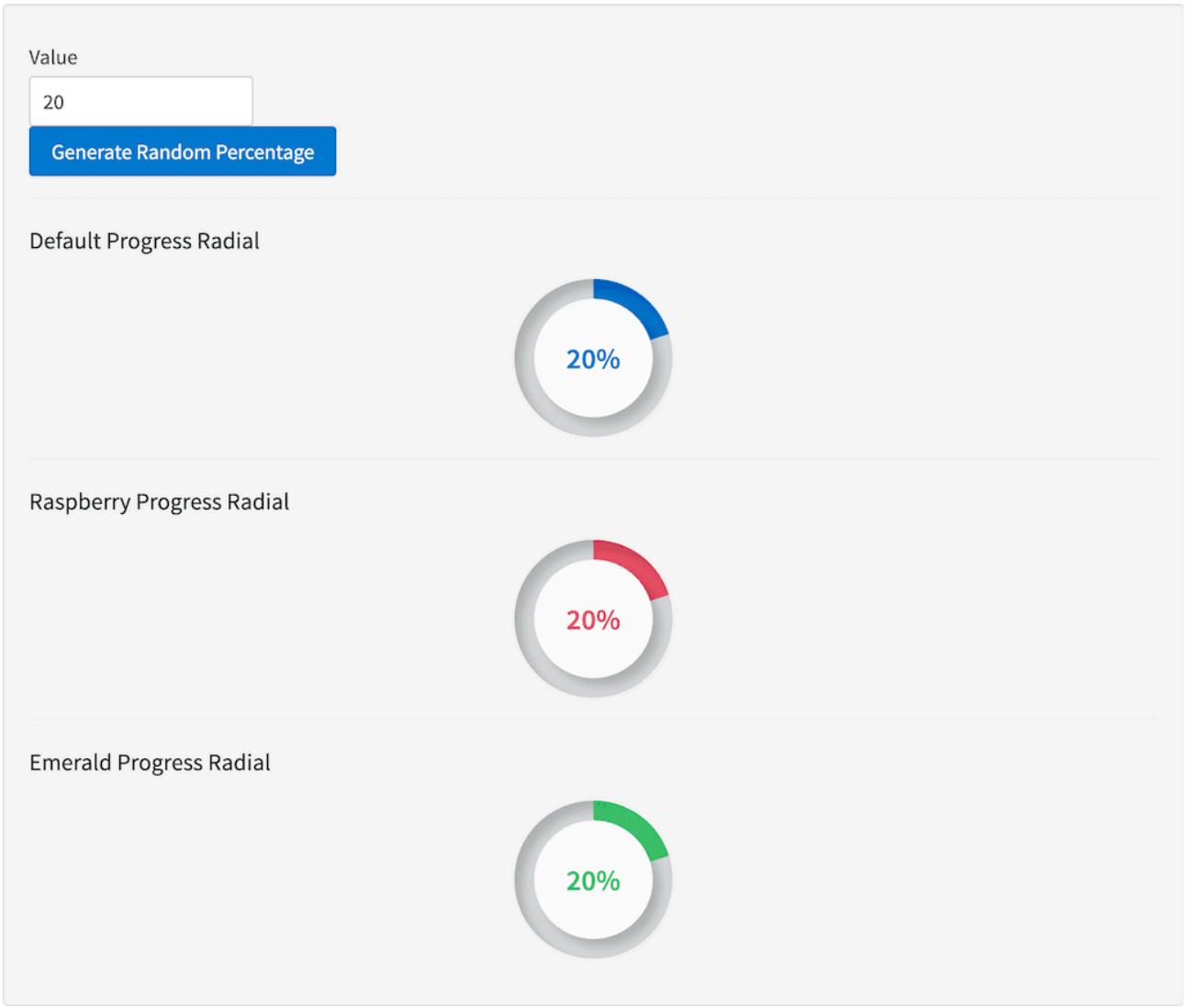
Example

Source

```
<label for="value">Value</label> <input id="value" type="number" placeholder="Value" ng-model="value"><br /> <button ng-click="generateNewNumber()" style="button">Generate Random Percentage</button> <hr /> <h3 id="example_default-progress-radial">Default Progress Radial</h3> <progress:radial percentage="value"> </progress:radial> <hr /> <h3 id="example_raspberry-progress-radial">Raspberry Progress Radial</h3> <progress:radial percentage="value" fill-class="tk-stl-bg-raspberry" percentage-class="tk-stl-text-raspberry"></progress:radial> <hr /> <h3 id="example_emerald-progress-radial">Emerald Progress Radial</h3> <progress:radial percentage="value" fill-class="tk-stl-bg-emerald" percentage-class="tk-stl-text-emerald"></progress:radial>
```

```
angular.module('lightning') .controller('ExampleController', ['$scope','$timeout', function($scope,$timeout) { $scope.value = 20; $scope.generateNewNumber = function(){ $scope.value = Math.floor(Math.random() * (100 - 0 + 1)); }; }]);
```

Result


Value
20
Generate Random Percentage
Default Progress Radial
20%
Raspberry Progress Radial
20%
Emerald Progress Radial
20%

query:custom

Description

The custom parameter directive is to be used within the *Query tag* to add custom query parameters.

Adding custom parameters

```
<search:query parameters="*" var="query">
    <query:custom name="facet.range" value="year"></query:custom>
    <query:custom name="facet.mincount" value="1"></query:custom>
</search:query>
```

Usage

as element:

```
<query:custom name="{string}" [value="{string}"]> </query:custom>
```

Parameters

Param	Type	Details
name	**string**	Name of the custom parameter.
value *(optional)*	**string**	Value of the custom parameter.

query:filter

Description

The filter directive is to be used within the *Query tag*.

Filter

Remove All Filters with Field Name

```
<search:query parameters="*" var="query">
    <query:filter field="company" action="remove""></query:filter>
</search:query>
```

Add Filter

```
<search:query parameters="*" var="query">
    <query:filter field="company" value="twigkit"></query:filter>
</search:query>
```

Remove Filter

```
<search:query parameters="*" var="query">
    <query:filter field="company" value="twigkit" action="remove"></query:filter>
</search:query>
```

Range Filter

Add Filter

```
<search:query parameters="*" var="query">
    <query:filter field="age" from="25" to="50" display="25 to 50"></query:filter>
</search:query>
```

Remove Filter

```
<search:query parameters="*" var="query">
    <query:filter field="age" from="25" to="50" display="25 to 50" action="remove">
</query:filter>
</search:query>
```

Usage

as element:

```
<query:filter field="{string}" [value="{string}"] [from="{Number|Date}"] [to="
{Number|Date}"] [display="{string}"] [optional="{string}"] [exclude="{boolean}"]
[action="{string}"] [hidden="{boolean}"] [match="{string}"] [type="{string}"]>
</query:filter>
```

Parameters

Param	Type	Details
field	**string**	The field the filter applies to.
value *(optional)*	**string**	The value to apply to the field
from *(optional)*	**Number]link:[Date**	The from value for the range filter.
to *(optional)*	**Number]link:[Date**	The to value for the range filter.
display *(optional)*	**string**	The display value for the range filter.
optional *(optional)*	**string**	If the filter is optional (or) (field or value)
exclude *(optional)*	**boolean**	If the filter should exclude values matching this filter.
action *(optional)*	**string**	Whether to remove the filter or add the filter. (Default:'add')
hidden *(optional)*	**boolean**	Whether the filter should be hidden (visible in breadcrumbs and URLs) or not.
match *(optional)*	**string**	Determines whether to match the value of the filter exactly, or partially, with either left or right wildcards. Accepted values are 'left_partial' or 'partial' or 'right_partial' or 'exact'.
type *(optional)*	**string**	Explicitly set the type of the filter, 'string' or 'number'. A string filter value will be wrapped in double quotes.

query:suggestions

Description

Query Suggestions

Usage

as element:

```
<query:suggestions completion-service="{string}" [action="{string}"] [service-url="{boolean}"] [show="{number}"] [title="{string}"]> </query:suggestions>
```

Directive info

- This directive creates new scope.

Parameters

Param	Type	Details
completion-service	**string**	Which service or services (as a comma separated list) to use for as-you-type suggestions, as defined in the configuration files for services.
action *(optional)*	**string**	Search action path information, e.g. 'search/'.
service-url *(optional)*	**boolean**	URL of the web service. Default is '\{{context}}/twigkit/services/query-suggestion-service/.
show *(optional)*	**number**	The number of suggestions to show.
title *(optional)*	**string**	An optional title to display above the list

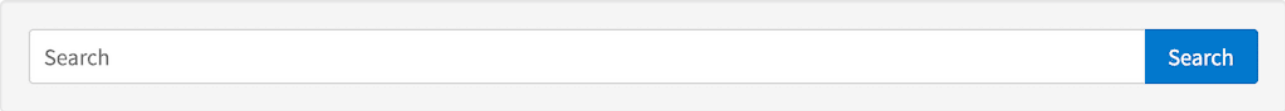
Example

Source

```
<search:box> <query:suggestions completion-service="services.suggestions.static"> </query:suggestions> </search:box>
```

```
angular.module('lightning') .controller('ExampleController',
['$scope','$timeout','ResponseService', function($scope,$timeout,ResponseService) {
}]);
```

Result



search:advanced

Description

Creates an advanced search form to allow the user to add multiple filters to a search query.

The following tags are commonly nested within a search advanced tag.

- *Search Box Tag*
- *Advanced Text Tag*
- *Advanced Date Range Tag*
- *Advanced Button Tag*

Usage

as element:

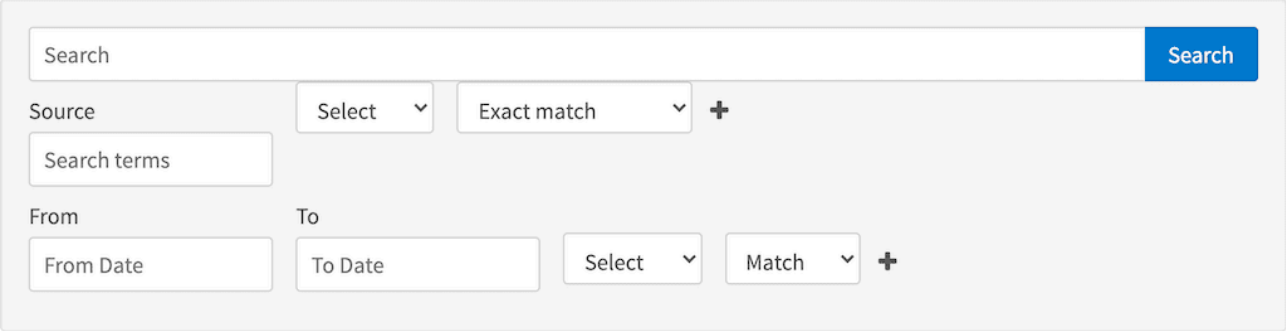
```
<search:advanced> </search:advanced>
```

Example

Source

```
<search:advanced> <search:box></search:box> <advanced:text
fields="source=Source,Material" label="Source"></advanced:text> <advanced:date-
range fields="source=Source,Material"></advanced:date-range> </search:advanced>
```

Result



search:alert

Description

Creates an alert box with the transcluded content

Usage

as element:

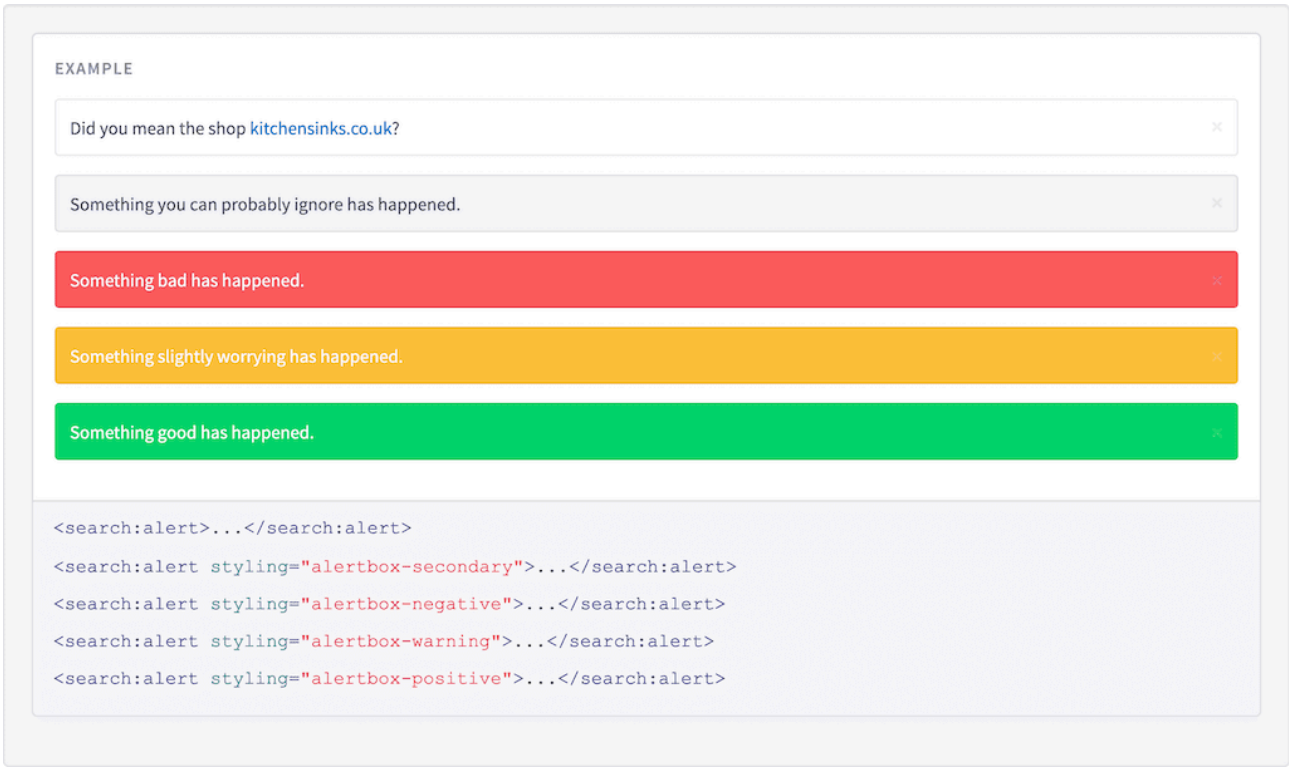
```
<search:alert> </search:alert>
```

Example

Source

```
<tk-example> <search:alert>Did you mean the shop <a
href="#">kitchensinks.co.uk</a>?</search:alert> <search:alert styling="alertbox-
secondary">Something you can probably ignore has happened.</search:alert>
<search:alert styling="alertbox-negative">Something bad has happened.
</search:alert> <search:alert styling="alertbox-warning">Something slightly
worrying has happened.</search:alert> <search:alert styling="alertbox-
positive">Something good has happened.</search:alert> </tk-example> <tk-code-
wrapper> <code hljs hljs-no-escape>&lt;search:alert&gt;...&lt;/search:alert&gt;
</code> <code hljs hljs-no-escape>&lt;search:alert styling="alertbox-
secondary"&gt;...&lt;/search:alert&gt;</code> <code hljs hljs-no-
escape>&lt;search:alert styling="alertbox-negative"&gt;...&lt;/search:alert&gt;
</code> <code hljs hljs-no-escape>&lt;search:alert styling="alertbox-
warning"&gt;...&lt;/search:alert&gt;</code> <code hljs hljs-no-
escape>&lt;search:alert styling="alertbox-positive"&gt;...&lt;/search:alert&gt;
</code> </tk-code-wrapper>
```

Result


EXAMPLE
Did you mean the shop kitchensinks.co.uk?
Something you can probably ignore has happened.
Something bad has happened.
Something slightly worrying has happened.
Something good has happened.
<search:alert>...</search:alert>
<search:alert styling="alertbox-secondary">...</search:alert>
<search:alert styling="alertbox-negative">...</search:alert>
<search:alert styling="alertbox-warning">...</search:alert>
<search:alert styling="alertbox-positive">...</search:alert>

search:box

Description

The Header Search directive

Usage

as element:

```
<search:box response="{Object}" [query="{string}"] [action="{string}"]
[placeholder="{string}"] [button-label="{string}"] [delay="{number}"] [platform="
{string}"] [size="{string}"] [social="{boolean}"] [search-within-checkbox="
{boolean}"] [search-within-label="{boolean}"] [search-within-default-value="
{boolean}"] [min-length="{number}"] [pre-search="{function}"]> </search:box>
```

Parameters

Param	Type	Details
response	**Object**	A response object
query *(optional)*	**string**	The name of the query object
action *(optional)*	**string**	Page the search should take place on, results will not change automatically when an action is set.
placeholder *(optional)*	**string**	Placeholder for the search box
button-label *(optional)*	**string**	Label of the button.
delay *(optional)*	**number**	The delay before searching, not applicable if an action is set or instant search is being used. If a delay is greater then 1, results will load instantly on the page.
platform *(optional)*	**string**	The platform to use for social search
size *(optional)*	**string**	The size of the search box (tiny, small, large, huge, massive)
social *(optional)*	**boolean**	Whether to enable the social features of the search header. Defaults to false. If you require a customized *Bookmark List Tag* or *Saved Query List Tag* you can add your own inside the search box tag and these will be used instead.
search-within-checkbox *(optional)*	**boolean**	Show the 'search within' option as a checkbox. This enables the query to retain the filters already applied when checked. If not checked the search is reset when a query is entered and the form submitted. If you want this functionality but don't want a checkbox showing, you can use this attribute set to be false, and set `search-within-default-value` to be true - this uses a hidden field instead of a checkbox.

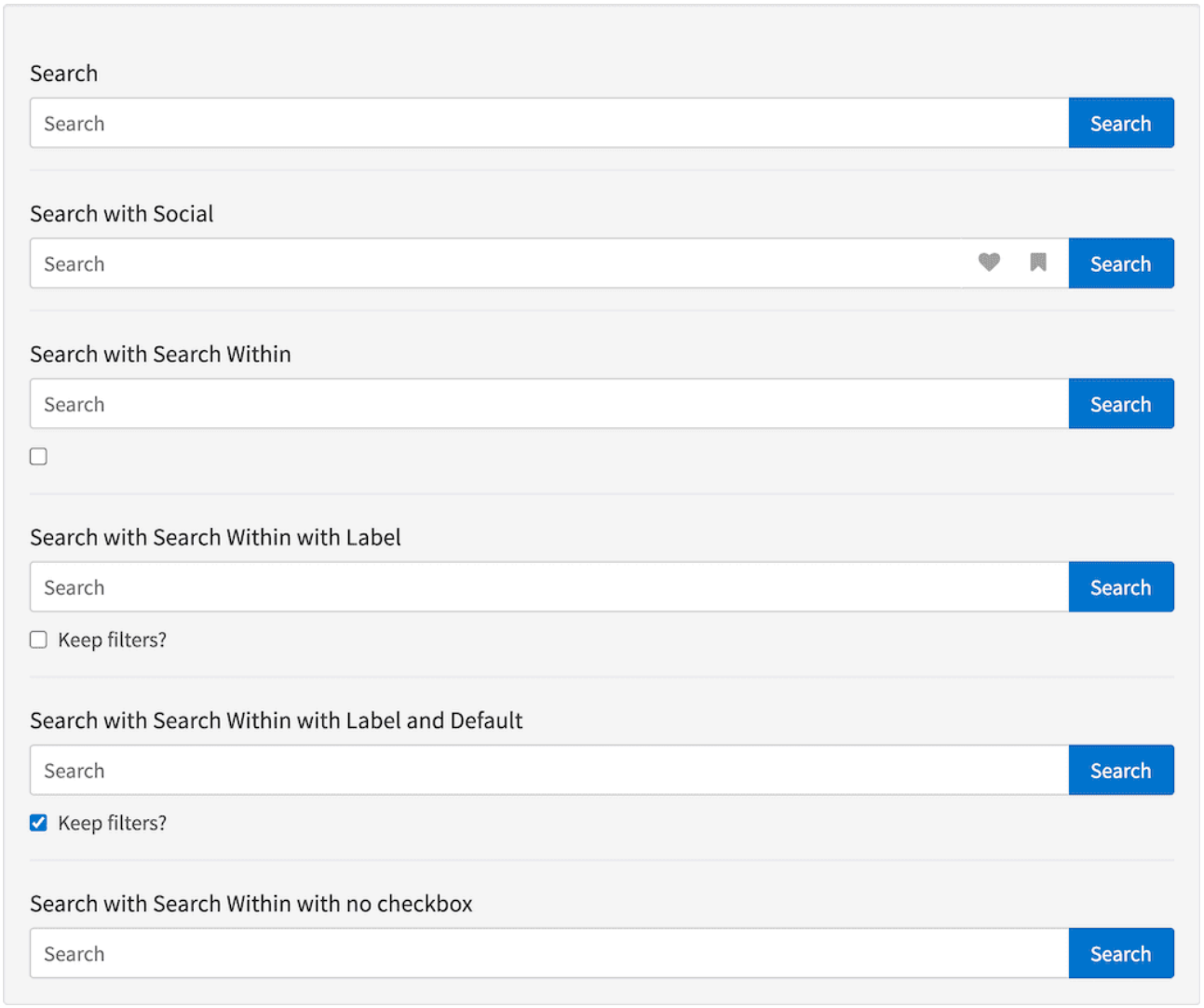
Param	Type	Details
search-within-label *(optional)*	**boolean**	The label for the 'search within' checkbox.
search-within-default-value *(optional)*	**boolean**	Whether to enable 'search within' by default. The default is option is 'false'.
min-length *(optional)*	**number**	The number of characters before a search is performed, only applicable for instant-results, query suggestions, advanced search or when a delay without an action. Default: 2
pre-search *(optional)*	**function**	Callback function that will be called on a search submit before the request is sent, receives the current URL state as a parameter that can be updated.

Example

Source

```
<h3 id="example_search">Search</h3> <search:box></search:box> <hr /> <h3
id="example_search-with-social">Search with Social</h3> <search:box social="true">
</search:box> <hr /> <h3 id="example_search-with-search-within">Search with Search
Within</h3> <search:box search-within-checkbox="true"></search:box> <hr /> <h3
id="example_search-with-search-within-with-label">Search with Search Within with
Label</h3> <search:box search-within-checkbox="true" search-within-label="Keep
filters?"></search:box> <hr /> <h3 id="example_search-with-search-within-with-
label-and-default">Search with Search Within with Label and Default</h3>
<search:box search-within-checkbox="true" search-within-label="Keep filters?"
search-within-default-value="true"></search:box> <hr /> <h3 id="example_search-
with-search-within-with-no-checkbox">Search with Search Within with no
checkbox</h3> <search:box search-within-checkbox="false" search-within-default-
value="true"></search:box>
```

Result


Search
Search
Search
Search with Social
Search
Search
Search with Search Within
Search
Search
Search with Search Within with Label
Search
Search
Keep filters?
Search with Search Within with Label and Default
Search
Search
Keep filters?
Search with Search Within with no checkbox
Search
Search

search:breadcrumbs

Description

The Breadcrumbs directive.

Usage

as element:

```
<search:breadcrumbs [response="{object}"] [query="{string}"] [group-by-field="
{boolean}"] [show-field-name="{boolean}"] [show-query="{boolean}"] [max-
characters="{number}"] [action="{string}"] [optional-label="{string}"] [date-
format="{string}"] [exclude="{string}"]> </search:breadcrumbs>
```

Directive info

- This directive creates new scope.

Parameters

Param	Type	Details
response *(optional)*	**object**	A response object
query *(optional)*	**string**	A query object
group-by-field *(optional)*	**boolean**	Group filters by field name (only available if query is not in grouping mode).
show-field-name *(optional)*	**boolean**	Whether to show field name as a part of the breadcrumb (only available when group by field is true), Default:true.
show-query *(optional)*	**boolean**	Whether to include the user query text as a breadcrumb. Default:true
max-characters *(optional)*	**number**	Limit display value to a certain number of characters, adding '…' if maxCharacters is exceeded.
action *(optional)*	**string**	Alternative page to submit the query parameters to - defaults to current page.
optional-label *(optional)*	**string**	Label to show when the Filter is optional.
date-format *(optional)*	**string**	Format to use for date values if no display value is specified. See #/api/lightning.filter:dateFormat[date format filter] for valid formats.
exclude *(optional)*	**string**	Comma separated list of fields to exclude, you can also use field=value to exclude only fields with the value provided.

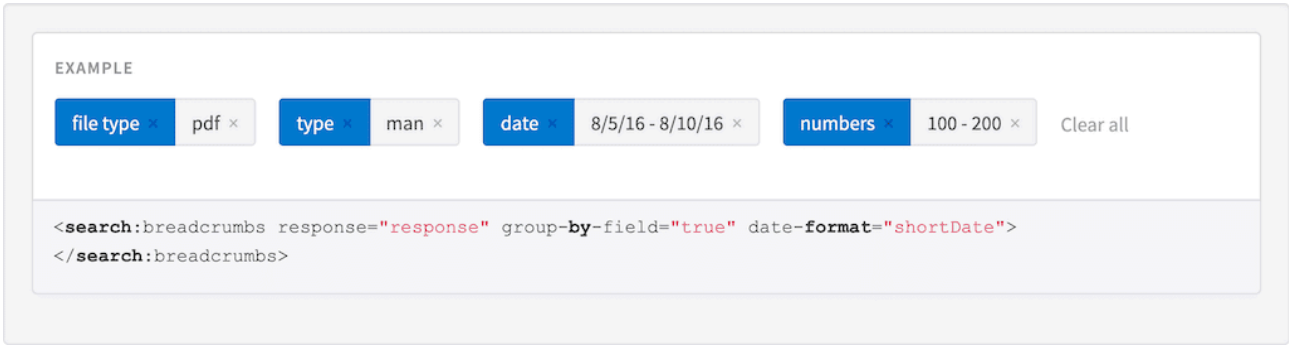
Example

Source

```
<tk-example> <search:breadcrumbs response="response" group-by-field="true" date-format="shortDate" exclude="problem"></search:breadcrumbs> </tk-example> <tk-code-wrapper> <code hljs><search:breadcrumbs response="response" group-by-field="true" date-format="shortDate"></search:breadcrumbs></code> </tk-code-wrapper>
```

```
angular.module('lightning') .controller('ExampleController', ['$scope','$timeout', function($scope,$timeout) { $scope.response = { page: 2, query: { rpp: 50, filters: [ { field:'file type', val:'pdf' }, { field:'type', val:'man' }, { val:{ act: { type: "range", from: "2016-08-05T12:50:30.660", to: "2016-08-10T12:50:40.509", fromExcluded: false, toExcluded:false }, dsp:'2016-08-05T12:50:30.660Z - 2016-08-10T12:50:40.509Z' }, field:'date' }, { val: { act: { type: "range", from: 100, to: "200", fromExcluded: false, toExcluded: false }, dsp: '100 - 200' }, field: 'numbers' }, { val: { act:'foo', dsp:'bar' }, field:'problem' } ] }, facets: {} } }]);
```

Result



search:conditional

Description

The search conditional directive

Usage

as element:

```
<search:conditional [test="{string}"] [response="{object}"]> </search:conditional>
```

Directive info

- This directive creates new scope.

Parameters

Param	Type	Details
test *(optional)*	**string**	The expression to test conditional logic for. Only render the contents tag if this evaluates to true.
response *(optional)*	**object**	A Response object to check, the tag will render the body if the Response is not null.

Example

Source

```
<h3 id="example_test-attribute">Test Attribute</h3> <search:conditional test="test1
> 100"> <p>False don't display</p> </search:conditional> <search:conditional
test="test1 >= 100"> <p>True! test1 is greater than or equal to 100</p>
</search:conditional> <search:conditional test="test2"> <p>True! test2 is true</p>
</search:conditional> <search:conditional test="test3 == 'Foo'"> <p>False don't
display</p> </search:conditional> <search:conditional test="test3 == 'foo'">
<p>True test3 does equal foo</p> </search:conditional> <hr/> <h3
id="example_response-attribute">Response Attribute</h3> <search:conditional
response="response"> <p>True! Response does exist.</p> </search:conditional>
<search:conditional response="response2"> <p>False! Response2 does not exist.</p>
</search:conditional>
```

```
angular.module('lightning') .controller('ExampleController',
['$scope','ResponseService', function($scope,ResponseService) { $scope.test1 = 100;
$scope.test2 = true; $scope.test3 = 'foo' ResponseService.setResponse('response',
{foo:'bar'}); }]);
```

Result

Test Attribute

True! test1 is greater than or equal to 100

True! test2 is true

True test3 does equal foo

Response Attribute

True! Response does exist.

search:facet

Description

The facet directive displays an individual facet. Either a response object and facet name must be passed to it, or a facet object.

Search Within Facet

To enable search within facets the following attributes must be set on the facet `search-enabled="true"`. The following attributes can be set on either the facet-list tag or the facet tag to enable search within `platform` and `query`.

If using a string field such as `facet_field_s` then you should add the attribute `facet-field` to your facet tag with the tokenized version of that field eg. `facet-field="facet_field_t"`, for more information see the `search-enabled` and `facet-field` attributes.

Multi-term facet queries

A field named `location_facet` might have several facets, such as 'AGL Centre', 'Airport City', and 'Airport Road'. When you search for 'AGL c', you want to see 'AGL Centre' in the results.

When conducting partial searches on multi-term facets, tokenize the searched field using `solr.KeywordTokenizerFactory`.

Lower-case transformation is crucial for case-insensitive matching. You can use the built-in `lowercase` field type in Fusion:

```
<fieldType name="lowercase" class="solr.TextField" positionIncrementGap="100">
<analyzer> <tokenizer class="solr.KeywordTokenizerFactory"/> <filter
class="solr.LowerCaseFilterFactory"/> </analyzer> </fieldType>
```

This applies a tokenizer and a lower-case filter to get partial and case-insensitive string matches when searching within facets.

Usage

as element:

```
<search:facet [response="{string}"] [facet-name="{string}"] [facet="{expression}"]
[title="{string}"] [show="{string}"] [show-more="{string}"] [max-characters="
{string}"] [select="{string}"] [platform="{string}"] [query="{string}"] [search-
enabled="{boolean}"] [wildcard="{string}"] [facet-field="{string}"] [collapsible="
{boolean}"] [show-empty="{boolean}"] [collapsed-by-default="{boolean}"] [show-
count="{boolean}"] [action="{string}"] [count-number="{string}"] [date-format="
{string}"] [measure="{string}"] [when-field="{string}"] [exclude="{boolean}"]>
</search:facet>
```

Directive info

- This directive creates new scope.

Parameters

Param	Type	Details
response *(optional)*	**string**	The name of the response to use.
facet-name *(optional)*	**string**	Facet name to use from the response object
facet *(optional)*	**expression**	A facet object to display
title *(optional)*	**string**	The title of the facet to display, defaults to the name of the facet
show *(optional)*	**string**	The number of filters to show by default
show-more *(optional)*	**string**	The number of extra filters to display when the show more link is clicked
max-characters *(optional)*	**string**	Limit display value to a certain number of characters, adding '…' if max-characters is exceeded.
select *(optional)*	**string**	How to render each FacetFilter. Valid values: • **click** User selects a filter by clicking the value; **(default)** • **multiselect** Supports selecting multiple filters that get combined with an 'OR'. • **toggle** Only allows one filter to be selected at a time but shows unselected filters and counts. • **multiselect-instant** Supports selecting multiple filters that get combined with an 'OR', filters will be applied automatically.

Param	Type	Details
platform *(optional)*	**string**	The name of the platform to use for search within the facet.
query *(optional)*	**string**	Nome of the query to be used.
search-enabled *(optional)*	**boolean**	Whether to enable search on the facet. If no query or platform is defined on the facet or facet-list it will enable the user to search through the filters that have been returned in the response. If a platform and query is defined on the facet or facet-list then it will search for filters in the search index. (Requires a tokenized field, if not using a tokenized field, if not using a tokenized field please see facet-field).
wildcard *(optional)*	**string**	If search-enabled is true you can set wildcard rules for this search. Valid values are: left: This adds a wildcard prefix to facet search terms, for example "*term*". *right: This adds a wildcard suffix to facet search terms, for example* "*term*". both (default): This adds both a wildcard prefix and suffix to facet search terms, for example "*term*". none: This adds no wildcards to the facet search term (only exact matches).
facet-field *(optional)*	**string**	When this option is provided, the user will have the option to search values within the facet using the named field to search against. The named field should be tokenized. Example usage is if you are using facet_field_s (a string facet/field) as your facet you need to provide the tokenized version of that field for search against, e.g. facet_field_t.
collapsible *(optional)*	**boolean**	Set to true if the facet should be collapsible in the list of facets. *(default: false)*
show-empty *(optional)*	**boolean**	If true then the facet will also output empty filters where count is zero. *(default: false)*
collapsed-by-default *(optional)*	**boolean**	Set to true if this facet should be collapsed by default. *(default: false)*
show-count *(optional)*	**boolean**	Whether to add metadata with frequency counts etc. *(default: true)*

Param	Type	Details
action *(optional)*	**string**	Alternative Action to submit the query parameters to - defaults to current page.
count-number *(optional)*	**string**	Specify how the number representing the count should be formatted. Valid values: • **plain** Displays the number without any formatting. • **formatted** Formatted means that the number is comma formatted (2,000,000 for 2000000). **(default)** • **rounded** Rounded means the number is represented in a short format (2k for ~2000).
date-format *(optional)*	**string**	How to format the date. See #/api/lightning.filter:dateFormat[date format filter] for valid formats.
measure *(optional)*	**string**	What fact to use as an alternative to count as quantitative value for a filter.
when-field *(optional)*	**string**	Comma-separated list of field names. When one matches this facet representation will be used during an iteration.
exclude *(optional)*	**boolean**	On filter hover, displays an icon to allow the user to add an exclusion filter for the value. *(default: false)*

Example

Source

```
<search:facet-list response="response" styling="facet-list facet-list-step facet-list-wrappedheader facet-list-wrappedheader-noborder"> <search:facet facet-name="test1" title="First Pass Files" collapsible="true"></search:facet> <search:facet facet-name="test2" title="Second Pass Files"></search:facet> <search:facet facet-name="empty" title="Empty Facet"></search:facet> <search:facet facet-name="date" title="Date Facet With Switches" date-format="fullDate" select="multiselect" styling="switches"></search:facet> <search:facet facet-name="date2" title="Date Facet With Checkboxes" date-format="fullDate" select="multiselect"></search:facet> <search:facet facet-name="date3" title="Date Facet With Format" date-format="dd/MM/yyyy"></search:facet> </search:facet-list> <search:facet-list response="response" styling="facet-list facet-list-step facet-list-wrappedheader facet-list-wrappedheader-noborder" facet-names="*"> <search:facet facet-name="date3" title="Date Facet With Format" date-format="dd/MM/yyyy"></search:facet> <search:facet></search:facet> </search:facet-list> <h5 id="example_source_show-count---false">Show Count - false</h5> <search:facet-list response="response" styling="facet-list facet-list-step facet-list-wrappedheader facet-list-wrappedheader-noborder" facet-names="*"> <search:facet facet-name="date3" title="Date Facet With Format" date-format="dd/MM/yyyy" show-count="false"></search:facet> <search:facet show-count="false"></search:facet> </search:facet-list>
```

```
angular.module('lightning') .controller('ExampleController', ['$scope','$timeout','ResponseService', function($scope,$timeout,ResponseService) { $scope.response = { page: 2, query: { rpp: 50 }, facets: { test1: { filters: [ { val: { dsp: 'Day 1', act: 'Day 1', }, count: 100, }, { val: { dsp: 'Day 2', act: 'Day 2', }, count: 200, } ], display:'Test 1' }, test2: { filters: [ { val: { dsp: 'Day 1', act: 'Day 1', }, count: 10, }, { val: { dsp: 'Day 2', act: 'Day 2', }, count: 20, } ], display:'Test 2' }, empty: { filters: [] }, date: { filters: [ { val: { dsp: 1288323623006, act: 1288323623006, }, count: 10, }, { val: { dsp: 1467882419615, act: 1467882419615, }, count: 20, } ], 'max-count':20, display:'Date' }, date2: { filters: [ { val: { dsp: 1288323623006, act: 1288323623006, }, count: 10, }, { val: { dsp: 1467882419615, act: 1467882419615, }, count: 20, } ], 'max-count':20, display:'date2' }, date3:{ filters:[ { val:{ act: { type: "range", from: "2016-08-05T12:50:30", to: "2016-08-05T12:50:40", fromExcluded: false, toExcluded:false }, dsp:'2016-08-05T12:50:30Z' }, count:10 } ], 'max-count':10, display:'date3' }, }, results: [] } ResponseService.setResponse('response', $scope.response); }]);
```

Result

First Pass Files

Day 1 100
Day 2 200

Second Pass Files

Day 1 10
Day 2 20

Date Facet With Switches

Thursday, October 28, 2010 10
Thursday, July 7, 2016 20

Date Facet With Checkboxes

Thursday, October 28, 2010 10
Thursday, July 7, 2016 20

Date Facet With Format

05/08/2016 10

Test 1

Day 1 100
Day 2 200

Test 2

Day 1 10
Day 2 20

Date

1288323623006 10
1467882419615 20

date2

1288323623006 10
1467882419615 20

Date Facet With Format

05/08/2016 10

Show Count - false

Test 1

Day 1

search:facet-list

Description

The facet list directive displays a list of facets.

Usage

as element:

```
<search:facet-list [response="{object}"] [styling="{string}"] [facet-names="
{string}"] [max-characters="{number}"] [platform="{string}"] [query="{Object}"]>
</search:facet-list>
```

Directive info

- This directive creates new scope.

Parameters

Param	Type	Details
response *(optional)*	**object**	A response object
styling *(optional)*	**string**	The style the facet list is to be rendered in. Accepted values 'shadow' and 'wrappedheader'
facet-names *(optional)*	**string**	The names of the facets to output. Can be specified as either a comma separated list or as a wildcard * for all available facets. In addition, you can use the - with a facet name to omit a specific facet (e.g. -cat will omit the cat facet).
max-characters *(optional)*	**number**	Limit display value to a certain number of characters, adding '…' if maxCharacters is exceeded. This will not break highlighting code.
platform *(optional)*	**string**	The name of the platform to use for search within the facet.
query *(optional)*	**Object**	Query to be used.

Example

Source

```
<h3 id="example_styling">Styling</h3> <h4 id="example_styling_simple">Simple</h4>
<search:facet-list response="response" facet-names="*"></search:facet-list> <h4
id="example_styling_wrapped-header">Wrapped Header</h4> <search:facet-list
response="response" styling="wrappedheader" facet-names="*"></search:facet-list>
<h4 id="example_styling_indented">Indented</h4> <search:facet-list
response="response" styling="indented" facet-names="*"></search:facet-list> <h4
id="example_styling_shadow">Shadow</h4> <search:facet-list response="response"
styling="shadow" facet-names="*"></search:facet-list> <h4
id="example_styling_hidemetadata">Hidemetadata</h4> <search:facet-list
response="response" styling="hidemetadata" facet-names="*"></search:facet-list> <h4
id="example_styling_recessed">Recessed</h4> <search:facet-list response="response"
styling="recessed" facet-names="*"></search:facet-list> <h4
id="example_styling_light">Light</h4> <div class="tk-stl-blocksidebar-light">
<aside class="tk-bl-sidebar"> <search:facet-list response="response" facet-
names="*"></search:facet-list> </aside> </div> <h4
id="example_styling_dark">Dark</h4> <div class="tk-stl-blocksidebar-dark"> <aside
class="tk-bl-sidebar"> <search:facet-list response="response" facet-names="*">
</search:facet-list> </aside> </div>
```

```
angular.module('lightning') .controller('ExampleController',
['$scope','$timeout','ResponseService', function($scope,$timeout,ResponseService) {
$scope.response = { page: 2, query: { rpp: 50 }, facets: { test1: { id:'test1',
display:'test1', filters: [ { val: { dsp: 'Day 1', act: 'Day 1', }, count: 100, },
{ val: { dsp: 'Day 2', act: 'Day 2', }, count: 200, } ] }, test2: { id:'test2',
display:'test2', filters: [ { val: { dsp: 'Day 1', act: 'Day 1', }, count: 10, }, {
val: { dsp: 'Day 2', act: 'Day 2', }, count: 20, } ] }, empty: { id:'empty',
display:'Empty', filters: [] }, }, results: [] }
ResponseService.setResponse('response', $scope.response); }]);
```

Result

Styling

Simple

test1

Day 1	100
Day 2	200

test2

Day 1	10
Day 2	20

Wrapped Header

test1	
Day 1	100
Day 2	200
test2	
Day 1	10
Day 2	20

Indented

test1

Day 1	100
Day 2	200

test2

Day 1	10
Day 2	20

Shadow

test1

Day 1	100
Day 2	200

test2

Day 1	10
Day 2	20

Hidemetadata

test1

Day 1
Day 2

test2

Day 1
Day 2

Recessed

test1

Day 1 100
Day 2 200

test2

Day 1 10
Day 2 20

Light

TEST1

Day 1 100
Day 2 200

TEST2

Day 1 10
Day 2 20

Dark

TEST1

Day 1 100
Day 2 200

TEST2

Day 1 10
Day 2 20

search:field

Description

The Field directive.

Usage

as element:

```
<search:field name="{string}" [result="{object}"] [query="{object}"] [max-characters="{string}"] [date-format="{string}"] [number-format="{string}"] [url="{string}"] [urlfield="{string}"] [target="{string}"] [action="{string}"] [default-value="{string}"] [label="{string}"] [value-element="{string}"] [label-element="{string}"] [prefix="{string}"] [suffix="{string}"] [show-more="{string}"] [show-less="{string}"] [show="{number}"] [show-more-values="{number}"] [separator="{string}"] [url-prefix="{string}"] [url-suffix="{string}"] [filter-optional="{string}"]> </search:field>
```

Directive info

- This directive creates new scope.

Parameters

Param	Type	Details
name	**string**	Name of the field to be rendered from the result
result *(optional)*	**object**	The result containing the field to be rendered
query *(optional)*	**object**	Providing a Query object means that the Value will be displayed as a hyperlink that submits the Query again with this field and its value as a Filt
max-characters *(optional)*	**string**	Limit display value to a certain number of characters, adding '…' if maxCharacters is exceeded.
date-format *(optional)*	**string**	How to format the date. See #/api/lightning.filter:dateFormat[date format filter] for valid formats.
number-format *(optional)*	**string**	Format string for the numbers used using the Java DecimalFormat pattern. *(default: ",0.")*
url *(optional)*	**string**	URL to redirect to when the field is clicked;
urlfield *(optional)*	**string**	Field containing a URL to redirect to when the field is clicked;
target *(optional)*	**string**	Set an HTML target attribute for the anchor tag. For example use _blank to have the generated link open the results in a new window. (Not applicable to query)
action *(optional)*	**string**	Alternative page to submit the query parameters to - defaults to current page. If a 'url' attribute is provided the action is ignored and query parameters are not included in the request.
default-value *(optional)*	**string**	Default value to show when no value is present in the index.
label *(optional)*	**string**	A label to use instead of the default value provided by the Platform. If this attribute is set to an empty String (“”) then the label container HTML is omitted from the output.
value-element *(optional)*	**string**	The HTML element to use for the value (Default p). *(default: "p")*

Param	Type	Details
label-element *(optional)*	**string**	The HTML element to use for the label (Default p) *(default: "p")*
prefix *(optional)*	**string**	Text to prefix index values with (not applied to default value). E.g. if the index field 'price' contains the value '10'. A prefix="US$" would result in 'US$10' would be displayed.
suffix *(optional)*	**string**	Text to append to index values (not applied to default value). E.g. if the index field 'weight' contains the value '10'. A suffix=" kg." would result in '10 kg.' being displayed as the value.
show-more *(optional)*	**string**	If the display value is limited by the max-characters, a show more link will be shown. The value of this attribute will be used as the label. *(default: "Show more")*
show-less *(optional)*	**string**	If the display value is limited by the max-characters, a show less link will be shown. The value of this attribute will be used as the label. *(default: "Show less")*
show *(optional)*	**number**	Number of values to show if the field is multivalued. *(default: -1)*
show-more-values *(optional)*	**number**	Set to the number of results you'd like the user to see after clicking the show more link. Set to `-1` to show all the remaining values for the field.
separator *(optional)*	**string**	Separator character or sequence to split a Field into multiple values. Setting the separator to ";" would turn the value "company1; company2; company3" into a list of 3 values.
url-prefix *(optional)*	**string**	Text to add as a prefix to the URL.
url-suffix *(optional)*	**string**	Text to add as a suffix to the URL.
filter-optional *(optional)*	**string**	If the fields filter is optional (or). Can be set to 'value', 'field' or 'none'. Default: none

Example

Source

```
<layout:grid> <layout:block> <h3 id="example_with-response">With Response</h3>
</layout:block> <search:result-list response="response" styling="cards-sm-1 cards-
md-2"> <search:result> <layout:block> <div class="tk-stl-card"> <search:field
name="name" styling="title" urlfield="url"></search:field> <search:field name="age"
styling="label-left" label="Age"></search:field> <search:field name="date"
styling="label-left" label="Created Date" date-format="dd/MM/yy"></search:field>
<search:field name="blank" styling="label-left" label="Contact Number" default-
value="No Contact Number on File."></search:field> <search:field name="number"
styling="label-left" label="Worth" number-format="£#,###.##"></search:field>
<search:field name="madeup" styling="label-left" label="Test"></search:field>
<search:field name="longstring" styling="label-left" label="Long String" max-
characters="40" show-more></search:field> </div> </layout:block> </search:result>
</search:result-list> </layout:grid> <layout:grid> <layout:block> <h3
id="example_with-results">With Results</h3> </layout:block> <search:result-list
results="results" styling="cards-sm-1 cards-md-2"> <search:result> <layout:block>
<div class="tk-stl-card"> <search:field name="name" styling="title" urlfield="url">
</search:field> <search:field name="age" styling="label-left" label="Age">
</search:field> <search:field name="date" styling="label-left" label="Created Date"
date-format="dd/MM/yy"></search:field> <search:field name="blank" styling="label-
left" label="Contact Number" default-value="No Contact Number on File.">
</search:field> <search:field name="number" styling="label-left" label="Worth"
number-format="£#,###.##"></search:field> <search:field name="madeup"
styling="label-left" label="Test"></search:field> </div> </layout:block>
</search:result> </search:result-list> </layout:grid>
```

```
angular.module('lightning') .controller('ExampleController',
['$scope','$timeout','ResponseService', function($scope,$timeout,ResponseService) {
$scope.response = { page: 2, query: { rpp: 50 }, hits:{ act:['long','1234'] },
facets: { }, results: [ { result: { fields: { name: { name: 'name', val: ['Testing
Tester'] }, url: { name: 'url', val: ['http://twigkit.com'] }, age: { name: 'age',
val: ['Long', 35] }, date: { name: 'date', val: ['2014-11-11T00:00:00'] }, blank:{
name:'blank', val:null, }, number:{ name:'number', val:[123456789] }, longstring:{
name:'longstring', val:['Lorem ipsum dolor sit amet, consectetur adipiscing elit,
sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim
veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo
consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum
dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident,
sunt in culpa qui officia deserunt mollit anim id est laborum.'] } }, }, }, {
result: { fields: { name: { name: 'name', val: ['Foo (Bar)'] }, url: { name: 'url',
val: ['http://twigkit.com'] }, age: { name: 'age', val: ['Long', 26] }, date: {
name: 'date', val: ['2015-11-11T00:00:00'] }, blank:{ name:'blank', val:null, },
number:{ name:'number', val:[-123456789] } }, }, }, ], } $scope.results =
$scope.response.results; ResponseService.setResponse('response', $scope.response);
}]);
```

Result

With Response

Testing Tester

Age	Long 35
Created Date	11/11/14
Contact Number	No Contact Number on File.
Worth	£123,456,789
Long String	Lorem ipsum dolor sit amet, consectetur ... show more

Foo (Bar)

Age	Long 26
Created Date	11/11/15
Contact Number	No Contact Number on File.
Worth	-£123,456,789

With Results

Testing Tester

Age	Long 35
Created Date	11/11/14
Contact Number	No Contact Number on File.
Worth	£123,456,789

Foo (Bar)

Age	Long 26
Created Date	11/11/15
Contact Number	No Contact Number on File.
Worth	-£123,456,789

search:fields

Description

A dropdown to select the fields which should be returned in the response, if no fields are selected all fields will be returned.

The fields tag should be wrapped in a *Layout Controls Tag*.

Usage

as element:

```
<search:fields [title="{string}"] [fields="{string}"] [hidden-fields="{string}"]>
</search:fields>
```

Directive info

- This directive creates new scope.

Parameters

Param	Type	Details
title *(optional)*	**string**	The title to display for the dropdown.
fields *(optional)*	**string**	A comma separated list of the fields that can be selected. Fields can be added with a display value like so actualFieldName=Display Field Name, anotherActual=My Field.
hidden-fields *(optional)*	**string**	A comma separated list of fields to be included with fields the user selects. Eg. id,firstname,lastname

Example

Source

```
<layout:controls> <search:fields fields="name=Name,current_address=Address,Gender">
</search:fields> </layout:controls>
```

Result

search:filter

Description

This has been deprecated see *Query Filter*.

The filter directive can be used to add a filter to the query.

Usage

as element:

```
<search:filter field="{string}" value="{string}" [optional="{boolean}"] [exclude="{boolean}"] [hidden="{boolean}"]> </search:filter>
```

Parameters

Param	Type	Details
field	**string**	The field the filter applies to.
value	**string**	The value to apply to the field
optional *(optional)*	**boolean**	If the filter is optional (or)
exclude *(optional)*	**boolean**	If the filter should exclude values matching this filter.
hidden *(optional)*	**boolean**	Whether the filter should be hidden (visible in breadcrumbs and URLs) or not.

search:filter-list

Description

Filter List Directive

Usage

as element:

```
<search:filter-list [facet="{expression}"] [facet-name="{string}"] [title="
{string}"] [show="{string}"] [show-more="{string}"] [max-characters="{string}"]
[select="{string}"] [show-empty="{boolean}"] [show-more-label="{string}"] [show-
less-label="{string}"] [show-count="{boolean}"] [action="{string}"] [order="
{string}"] [count-number="{string}"] [date-format="{string}"] [measure="{string}"]
[exclude="{boolean}"]> </search:filter-list>
```

Directive info

- This directive creates new scope.

Parameters

Param	Type	Details
facet *(optional)*	**expression**	A facet object to display
facet-name *(optional)*	**string**	Facet name to use from the response object
title *(optional)*	**string**	The title of the facet to display, defaults to the name of the facet
show *(optional)*	**string**	The number of filters to show by default
show-more *(optional)*	**string**	The number of extra filters to display when the show more link is clicked
max-characters *(optional)*	**string**	Limit display value to a certain number of characters, adding '…' if maxCharacters is exceeded.
select *(optional)*	**string**	How to render each FacetFilter. 'click': User selects a filter by clicking the value; 'multiselect': Supports selecting multiple filters that get combined with an 'OR'. 'toggle': Only allows one filter to be selected at a time but shows unselected filters and counts. 'multiselect-instant': Supports selecting multiple filters that get combined with an 'OR', filters will be applied automatically. Default: 'click'
show-empty *(optional)*	**boolean**	If true then the filter list will also output empty filters where count is zero. Default: false
show-more-label *(optional)*	**string**	Text to show on the link that shows more filters (expandable mode). Default: 'Show more'.

Param	Type	Details
show-less-label *(optional)*	**string**	Text to show on the link that shows fewer filters (expandable mode). Default: 'Show less'.
show-count *(optional)*	**boolean**	Whether to add metadata with frequency counts etc. Default: true
action *(optional)*	**string**	Alternative Action to submit the query parameters to - defaults to current page.
order *(optional)*	**string**	Specify a sort order for this filter list. Allowed values are: 'alphabetical', '-alphabetical', '-count' and 'count'. Default is whatever the search engine provides.
count-number *(optional)*	**string**	Specify how the number representing the count should be formatted. Optional values are 'plain', 'formatted' and 'rounded'. Rounded means the number is represented in a short format (2k for ~2000). Formatted means that the number is comma formatted (2,000,000 for 2000000).Default: 'formatted'
date-format *(optional)*	**string**	How to format the date. See #/api/lightning.filter:dateFormat[date format filter] for valid formats.
measure *(optional)*	**string**	What fact to use as an alternative to count as quantitative value for a filter.
exclude *(optional)*	**boolean**	On filter hover, displays an icon to allow the user to add an exclusion filter for the value. Default: false

search:keyword

Description

Search keyword tag allows you to add the ability for a user to filter the search based on the keyword they enter and the value they select.

Usage

as element:

```
<search:keyword keyword="{string}" facet="{string}" [facet-field="{string}"] [description="{string}"] [date="{string}"] [date-format="{string}"] [date-display-format="{string}"]> </search:keyword>
```

Directive info

- This directive creates new scope.

Parameters

Param	Type	Details
keyword	**string**	The keyword the user types
facet	**string**	The facet the keyword is associated with
facet-field *(optional)*	**string**	The facet the keyword is associated with
description *(optional)*	**string**	The description to display next to the keyword
date *(optional)*	**string**	Type of date (on, before, after. Default: on)
date-format *(optional)*	**string**	Date format to use for the datapicker. Default yyyy-MM-ddTHH:mm:ss
date-display-format *(optional)*	**string**	Date format to use for the datapicker display values. Default yyyy-MM-ddTHH:mm:ss

Example

Source

```
<search:advanced keywords="keywords" response="response" platform="platform">
<search:keyword keyword="from" facet="Start_Date" date="after"
description="Creation date"></search:keyword> <search:keyword keyword="type"
facet="Study_Types" description="Type of study"></search:keyword> <search:keyword
keyword="gender" facet="Gender" description="Patient gender"></search:keyword>
<search:keyword keyword="Conditions"></search:keyword> <search:keyword
keyword="Recruitment"></search:keyword> <search:keyword keyword="Sponsors"
facet="Sponsor_Collaborators"></search:keyword> <instant:result-list
platform="platform"> <search:result> <a ng-href="/#/result/{{ result.fields.id.val
| actualValue }}">{{ result.fields.Title.val | displayValue }}</a> </search:result>
</instant:result-list> </search:advanced>
```

search:load

Description

The load directive sends an id or a query to the twigkit API and stores the result on a scoped variable and in the *Response Service*

Usage

as element:

```
<search:load var="{string}" platform="{string}" id="{string}" query="{string}"
[error-page="{string}"]> </search:load>
```

Directive info

- This directive creates new scope.

Parameters

Param	Type	Details
var	**string**	The name of the variable to store the document in.
platform	**string**	The name of the platform to retrieve the document from.
id	**string**	The ID of the document to retrieve. If both an ID and query are present ID will take precedence.
query	**string**	The name of the query object. If both an ID and query are present ID will take precedence.
error-page *(optional)*	**string**	Error page to redirect to when Platform throws an exception.

search:message-list

Description

Output Messages and QuerySuggestions such as spellchecking or synonyms.

Custom messages within a message-list **must be wrapped in another html tag!** - see *Example Custom*

Usage

as element:

```
<search:message-list response="{string}" [type="{string}"] [title="{string}"]
[first="{number}"] [last="{number}"]> </search:message-list>
```

Directive info

- This directive creates new scope.

Parameters

Param	Type	Details
response	**string**	The name of the response containing the messages to display.
type *(optional)*	**string**	The type of Message to display. Default is to include all messages irrespective of type.
title *(optional)*	**string**	Add a title to appear with this element.
first *(optional)*	**number**	First message to output from this result set (1 based).
last *(optional)*	**number**	Last message to output from this result set.

Example

Source

```
<h3 id="example_all-messages">All messages</h3> <tk-example> <search:message-list response="response" title="My Messages"></search:message-list> </tk-example> <tk-code-wrapper> <code hljs language="html"> <search:message-list response="response" title="My Fuzzy Results"></search:message-list> </code> </tk-code-wrapper> <hr /> <h3 id="example_only-messages-of-selected-type">Only messages of selected type</h3> <tk-example> <search:message-list response="response" type="fuzzy-results" title="My Fuzzy Results"></search:message-list> </tk-example> <tk-code-wrapper> <code hljs language="html"> <search:message-list response="response" type="fuzzy-results" title="My Fuzzy Results"></search:message-list> </code> </tk-code-wrapper> <hr /> <h3 id="example_first-and-last-messages">First and Last Messages</h3> <tk-example> <search:message-list response="response" first="1" last="2" title="My Messages"></search:message-list> </tk-example> <tk-code-wrapper> <code hljs language="html"> <search:message-list response="response" first="1" last="2" title="My Messages"></search:message-list> </code> </tk-code-wrapper> <hr /> <h3 id="example_custom">Custom</h3> <tk-example> <search:message-list response="response" title="My Fuzzy Results"> <p><strong>{{message.type}}</strong> - <span ng-bind-html="message.value"></span> <small>({{message.count}})</small></p> </search:message-list> </tk-example> <tk-code-wrapper> <code hljs language="html"> <search:message-list response="response" title="My Fuzzy Results"> <p> <strong> {{message.type}}</strong> - <span ng-bind-html="message.value"></span> <small> ({{message.count}})</small> </p> </search:message-list> </code> </tk-code-wrapper>
```

```
angular.module('lightning') .controller('ExampleController', ['$scope','ResponseService', function($scope,ResponseService) { $scope.response = { "query": { "filters": [], }, "hits": { "act": [ "Long", 0 ] }, "page": 2, "results": [], "facets": {}, "facts": {}, "time": 19, "messages": [ { "type": "fuzzy-results", "value": "<em>Lorem</em> ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.", "count": 20 }, { "type": "not-fuzzy-results", "value": "Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur.", "count": 100 }, { "type": "fuzzy-results", "value": "Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat.", "count": 0 } ] } ResponseService.setResponse('response',$scope.response); }]);
```

Result

All messages

EXAMPLE

My Messages

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.

Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur.

Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat.

```
<search:message-list response="response" title="My Fuzzy Results"></search:message-list>
```

Only messages of selected type

EXAMPLE

My Fuzzy Results

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.

Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat.

```
<search:message-list response="response" type="fuzzy-results" title="My Fuzzy Results">
</search:message-list>
```

First and Last Messages

EXAMPLE

My Messages

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.

Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur.

```
<search:message-list response="response" first="1" last="2" title="My Messages">
</search:message-list>
```

Custom

EXAMPLE

My Fuzzy Results

fuzzy-results - *Lorem* ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. (20)

not-fuzzy-results - Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. (100)

fuzzy-results - Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. (0)

```
<search:message-list response="response" title="My Fuzzy Results">
    <p>
```

```
        <strong>{{message.type}}</strong> - <span ng-bind-html="message.value"></span> <small>
({{message.count}})</small>
    </p>
</search:message-list>
```

search:no-results

Description

The no results directive will display a message when no results are found on the response.

This tag should not be put in the `<search:result-list></search:result-list>`

Usage

as element:

```
<search:no-results response="{string}"> </search:no-results>
```

Directive info

- This directive creates new scope.

Parameters

Param	Type	Details
response	**string**	The response object containing the results

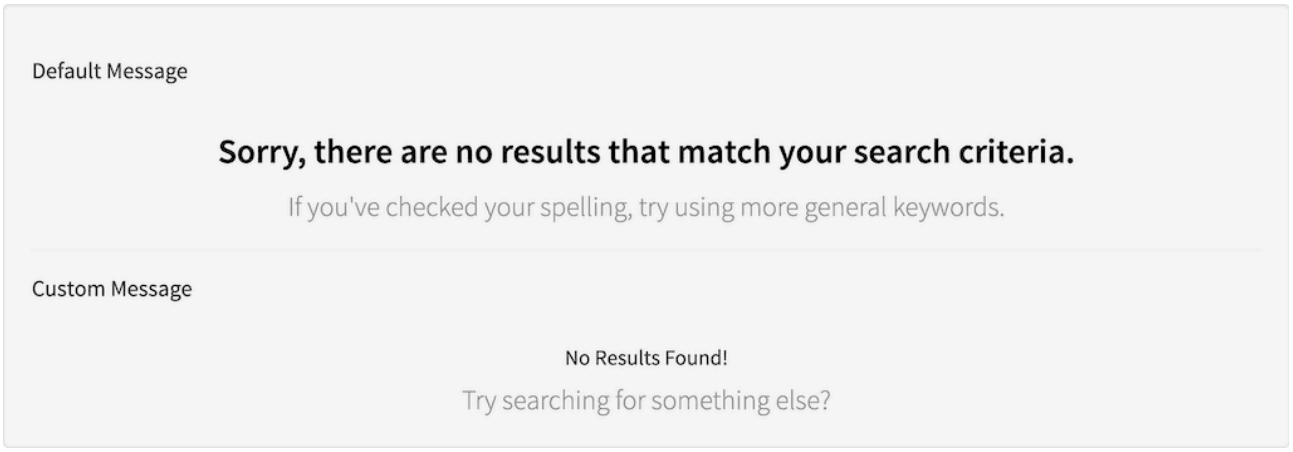
Example

Source

```
<h5 id="example_source_default-message">Default Message</h5> <search:result-list>
</search:result-list> <search:no-results response="response"></search:no-results>
<hr /> <h5 id="example_source_custom-message">Custom Message</h5> <search:result-
list></search:result-list> <search:no-results response="response"> <h4
id="example_no-results-found">No Results Found!</h4> <p>Try searching for something
else?</p> </search:no-results>
```

```
angular.module('lightning').controller('ExampleController', ['$scope',
'ResponseService', function($scope, ResponseService) { var response = {};
ResponseService.setResponse('response', response); }]);
```

Result


Default Message
Sorry, there are no results that match your search criteria.
If you've checked your spelling, try using more general keywords.
Custom Message
No Results Found!
Try searching for something else?

search:offcanvas-toggle

Description

Creates a toggle button to show/hide the side panel. Adds class to target element when directive is clicked.

Usage

as element:

```
<search:offcanvas-toggle target="{string}" [active-class="{string}"]>
</search:offcanvas-toggle>
```

Parameters

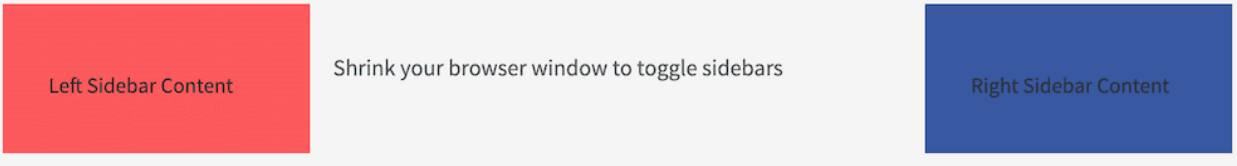
Param	Type	Details
target	**string**	The element to add the active class to
active-class *(optional)*	**string**	The active class, defaults to 'active'

Example

Source

```
<layout:grid> <layout:block tiny="1-4"> <layout:box> <search:offcanvas-toggle
target="#left-sidebar"></search:offcanvas-toggle> </layout:box> </layout:block>
<layout:block tiny="1-2"> <!-- this toggle will not be shown as no element with the
ID no-sidebar exists --> <search:offcanvas-toggle target="#no-sidebar">
</search:offcanvas-toggle> </layout:block> <layout:block tiny="1-4"> <layout:box>
<search:offcanvas-toggle target="#right-sidebar" class="right"></search:offcanvas-
toggle> </layout:box> </layout:block> </layout:grid> <layout:grid styling="row-
offcanvas"> <layout:block medium="1-4" drawer="left" id="left-sidebar"> <layout:box
class="tk-stl-bg-red"> <layout:sidebar> <h4 id="example_left-sidebar-content">Left
Sidebar Content</h4> </layout:sidebar> </layout:box> </layout:block> <layout:block
medium="1-2"> <layout:box> <p styling="text-align:center;">Shrink your browser
window to toggle sidebars</p> </layout:box> </layout:block> <layout:block
medium="1-4" drawer="right" id="right-sidebar"> <layout:box class="tk-stl-bg-blue">
<layout:sidebar> <h4 id="example_right-sidebar-content">Right Sidebar Content</h4>
</layout:sidebar> </layout:box> </layout:block> </layout:grid>
```

Result



search:pagination

Description

The pagination directive.

Usage

as element:

```
<search:pagination response="{string}" [max-pages="{number}"] [page-limit="
{number}"] [first-label="{string}"] [show-last="{boolean}"] [scroll-to="
{string|boolean}"]> </search:pagination>
```

Directive info

- This directive creates new scope.

Parameters

Param	Type	Details
response	**string**	The name of the response to use to generate the pagination.
max-pages *(optional)*	**number**	Maximum number of page links to output (default is 10).
page-limit *(optional)*	**number**	Maximum limit for which a pagination link is created (e.g. FAST ESP has a default offset limit of ~4000).
first-label *(optional)*	**string**	Label for the link to the first page. Default: '1'
show-last *(optional)*	**boolean**	Whether to show a link to the last page. Default: false
scroll-to *(optional)*	**string]link: [boolean**	The ID of an element to scroll to when the users selects a page. By default it will scroll to the top. You can disable this automatic scrolling by setting this attribute to `false`

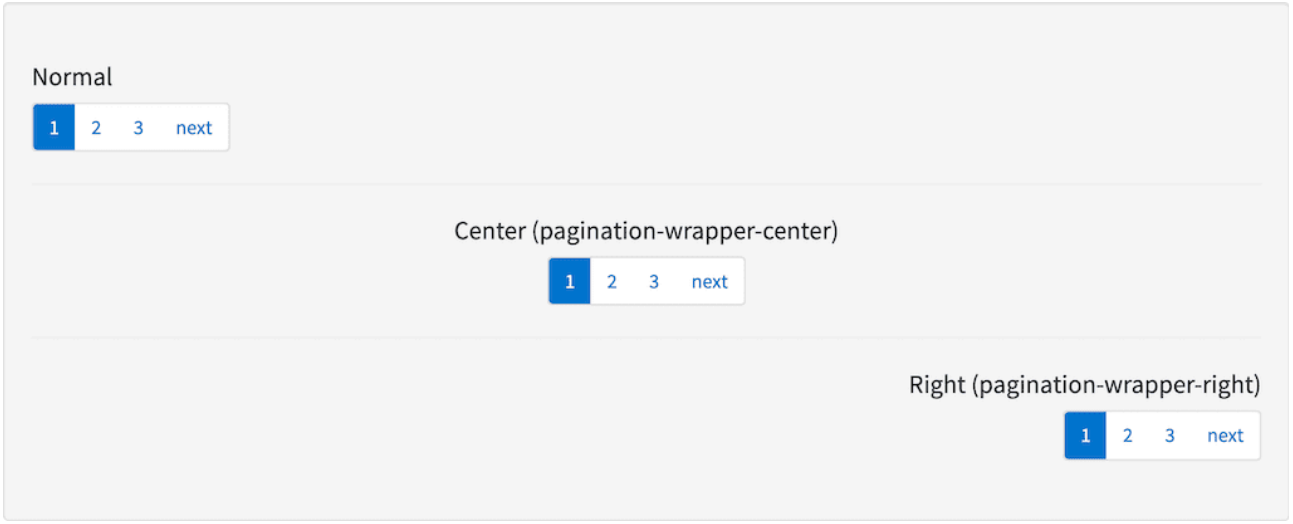
Example

Source

```
<h3 id="example_normal">Normal</h3> <search:pagination response="response" max-pages="3"></search:pagination> <hr /> <h3 styling="text-center" id="example_center">Center (pagination-wrapper-center)</h3> <search:pagination response="response" max-pages="3" styling="pagination-wrapper-center"></search:pagination> <hr /> <h3 style="text-align:right;" id="example_right">Right (pagination-wrapper-right)</h3> <search:pagination response="response" max-pages="3" styling="pagination-wrapper-right"></search:pagination>
```

```
angular.module('lightning').controller('ExampleController', ['$scope', 'ResponseService', function($scope, ResponseService) { var response = {query:{rpp:10},hits:{act:[0,22555]},page:1};
ResponseService.setResponse('response',response); }]);
```

Result



search:platform

Description

The platform directive creates an instance of platform for a given host and configuration.

Usage

as element:

```
<search:platform [var="{string}"] [host="{string}"] conf="{string}">
</search:platform>
```

Directive info

- This directive creates new scope.

Parameters

Param	Type	Details
var *(optional)*	**string**	The variable to assign the platform instance to.
host *(optional)*	**string**	The host and port name to use (default is to not define it).
conf	**string**	Platform configuration to use.

search:query

Description

The query directive constructs a query string and stores it in a scoped variable, *Query Filter's* can be added to add or remove filters from the query. You can also use the *Query Custom Tag* to add custom parameters to the query.

Usage

as element:

```
<search:query var="{string}" parameters="{string}" [from="{string}"] [query="
{string}"] [modified-query="{string}"] [type="{string}"] [results-per-page="
{string}"] [max-results="{string}"] [page="{string}"] [facets="{string}"] [fields="
{string}"] [sorts="{string}"] [view="{string}"] [filters="{string}"] [collection="
{string}"] [custom="{string}"]> </search:query>
```

Directive info

- This directive creates new scope.

Parameters

Param	Type	Details
var	**string**	The variable to assign the query to
parameters	**string**	The URL parameters to query with.
from *(optional)*	**string**	Name of a previously defined query to extend
query *(optional)*	**string**	Set the search query value.
modified-query *(optional)*	**string**	Specify an alternative query which will be submitted to the search engine in place of the query parameter supplied by the user (but can include user submitted parameter). To the user it will appear as the query parameter had been used, but the modifiedQuery will be submitted to the search engine.
type *(optional)*	**string**	Specifies how to interpret multiple query terms. Takes one of three values: any (boolean OR the terms), all (boolean OR/AND the terms), or adv (interpret using search platform-specific advanced query language).
results-per-page *(optional)*	**string**	The number of results to display per page.
max-results *(optional)*	**string**	The maximum number of results to evaluate. Defaults to -1 which means no upper limit.
page *(optional)*	**string**	Set the page of results to display. Default value is 1
facets *(optional)*	**string**	Specifies which facets to evaluate and return.
fields *(optional)*	**string**	A comma-separated list of fields which should be returned in the Response.

Param	Type	Details
sorts *(optional)*	**string**	Specifies fields and order for sorting. Sort order is specified by one of two characters: + for ascending and - for descending. To sort ascending by the field price, the sort would be specified as sorts="+price".
view *(optional)*	**string**	Set which search engine view to use. Behaviour will vary by installation.
filters *(optional)*	**string**	Specifies filters to apply, specified as an ampersand (&) separated String
collection *(optional)*	**string**	Specifies the collection you want to use.
custom *(optional)*	**string**	Specifies custom parameters to apply, specified as an ampersand (&) separated String

Example

Source

```
<search:query var="query" filters='zipcode["75216"]["75216"],bar["foo"]["foo"]' max-results="1"> <query:filter field="foo" value="bar"></query:filter> <query:custom name="foo" value="bar"></query:custom> </search:query> <search:query var="copyQuery" from="query"></search:query> <div ng-if="query != null"> <code>Original - {{query | setQueryParameter:'q':'foo'}}</code> </div> <code>Copy - {{copyQuery}}</code>
```

```
angular.module('lightning') .controller('ExampleController', ['$scope','$timeout', function($scope,$timeout) { }]); angular .module('lightning') .filter('setQueryParameter', setQueryParameter); setQueryParameter.$inject = ['$timeout']; function setQueryParameter($timeout) { return function (query, parameter, value) { if (query && parameter && value) { query[parameter] = value; return query; } else { return query } }; }
```

Result

```
Original - {"max":"1","filters":[{"exclude":false,"field":"zipcode","val":{"act":"75216","dsp":"75216\"],bar[\"f
Copy - {"max":"1","filters":[{"exclude":false,"field":"zipcode","val":{"act":"75216","dsp":"75216\"],bar[\"foo\"
```

search:response

Description

The response directive sends a query to the twigkit API and stores the response on a scoped variable and in the *Response Service*

Inside of the tag you can access the average response time (`$parent.averageResponseTime`) and the previous response time (`$parent.previousResponseTime`).

Usage

as element:

```
<search:response var="{string}" platform="{object}" query="{object}" [error-page="
{string}"] [on="{string}"]> </search:response>
```

Directive info

- This directive creates new scope.

Parameters

Param	Type	Details
var	**string**	The name of the variable to store the response in
platform	**object**	The platform to query against
query	**object**	A query object
error-page *(optional)*	**string**	Error page to redirect to when Platform throws an exception.
on *(optional)*	**string**	Event that is broadcasted for when the response should reload. Default: reloadSearch.

search:response-statistics

Description

The Response Statistics directive.

Usage

as element:

```
<search:response-statistics response="{object}" [number-format="{string}"]>
</search:response-statistics>
```

Directive info

- This directive creates new scope.

Parameters

Param	Type	Details
response	**object**	A response object
number-format *(optional)*	**string**	Format string for the numbers used using the Java DecimalFormat pattern. Default: ',0.'

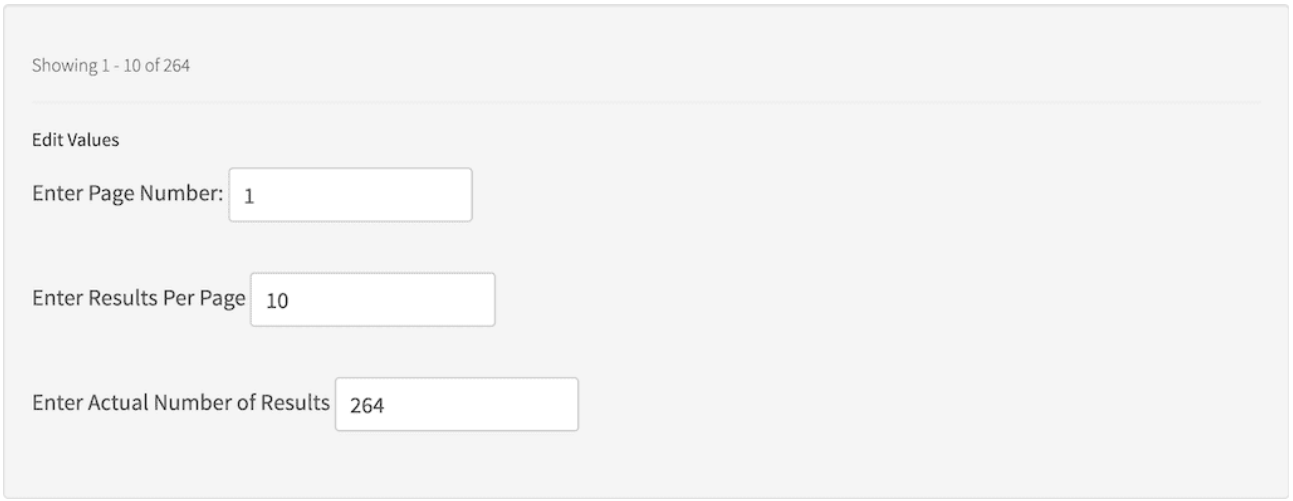
Example

Source

```
<search:response-statistics response="response"></search:response-statistics> <hr
/> <h6 id="example_source_edit-values">Edit Values</h6> <label>Enter Page Number:
<input ng-model='response.page' ng-change="reloadResponse()" /></label><br>
<label>Enter Results Per Page <input ng-model='response.query.rpp' ng-
change="reloadResponse()" /></label><br> <label>Enter Actual Number of Results
<input ng-model='response.hits.act[1]' ng-change="reloadResponse()" /></label><br>
```

```
angular.module('lightning') .controller('ExampleController', ['$scope',
'ResponseService', function($scope, ResponseService) { $scope.response = { hits:{
act:[264,264] }, page:1, query:{ rpp:10 } }
ResponseService.setResponse('response',$scope.response); $scope.reloadResponse =
function(){ var response = angular.copy($scope.response); $scope.response = {};
$scope.response = response;
ResponseService.setResponse('response',$scope.response); } }]);
```

Result



search:result

Description

The Result directive.

Usage

as element:

```
<search:result [result="{object}"] [when="{string}"] [equals="{string}"]>
</search:result>
```

Directive info

- This directive creates new scope.

Parameters

Param	Type	Details
result *(optional)*	**object**	A result object
when *(optional)*	**string**	The result will only be rendered if this field exists
equals *(optional)*	**string**	The result will only be rendered if this is the value of the when field

search:result-list

Description

The Result List directive.

Accessing result index

Within a search result you can access the index of the result by doing `{{$parent.index}}`. The index is zero-based so for result 1 the index will be 0.

Usage

as element:

```
<search:result-list response="{object}" results="{array}" [infinite-scroll="
{string}"] [query="{string}"] [platform="{string}"] [load-more="{boolean}"]>
</search:result-list>
```

Directive info

- This directive creates new scope.

Parameters

Param	Type	Details
response	**object**	A response object (Use response or results not both)
results	**array**	A list of results (Use response or results not both)
infinite-scroll *(optional)*	**string**	Whether infinite scrolling should be enabled (query and platform is required for this functionality to be enabled). Valid values: • **false** Disables infinite scrolling. **(default)** • **true** Enables infinite-scrolling in **window** mode (see below). • **window** When the window is scrolled to the bottom of the result list new results will be loaded. • **parent** When the parent element is scrollable when this element is scrolled to the bottom new results will load.
query *(optional)*	**string**	The name of the query object used for infinite scrolling or load more.
platform *(optional)*	**string**	The name of the platform object used for infinite scrolling or load more.
load-more *(optional)*	**boolean**	Whether to have a clickable link to load more results.

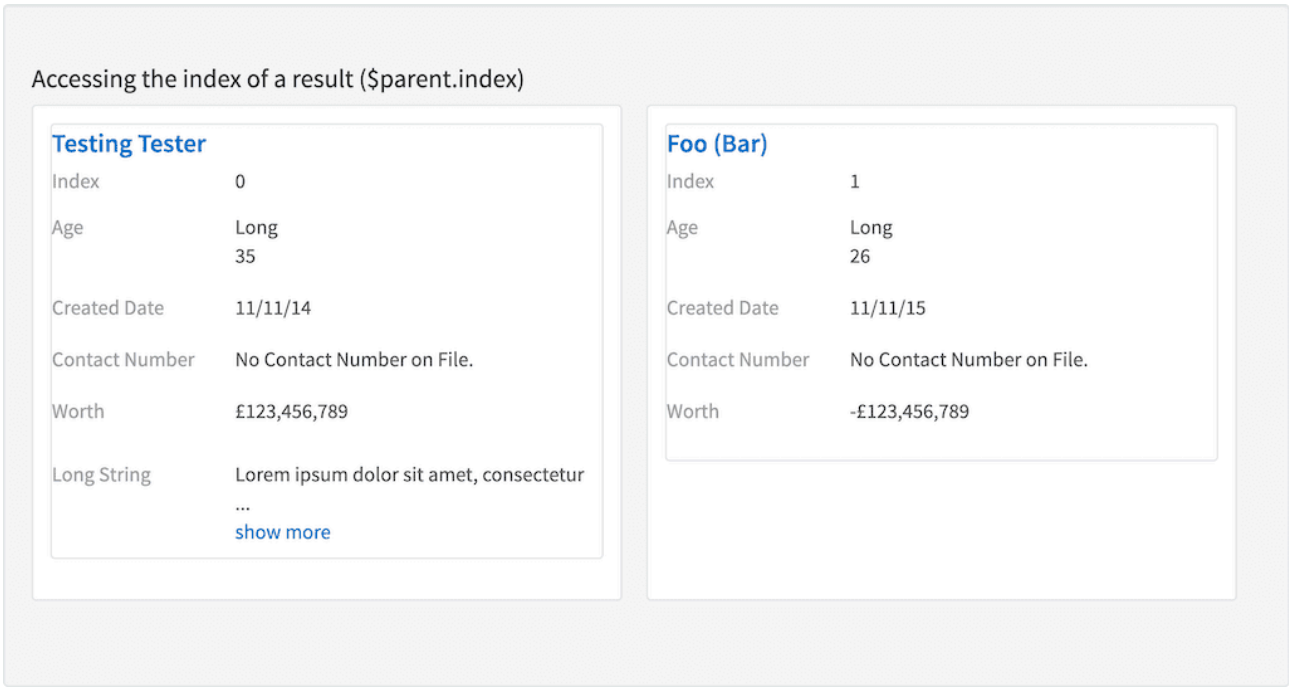
Example

Source

```
<layout:grid> <layout:block> <h3 id="example_accessing-the-index-of-a-result">Accessing the index of a result ($parent.index)</h3> </layout:block> <search:result-list response="response" styling="cards-sm-1 cards-md-2"> <search:result> <layout:block> <div class="tk-stl-card"> <search:field name="name" styling="title" urlfield="url"></search:field> <div class="field tk-rgrid-g tk-stl-label-left"> <p class="tk-rgrid-u-1 tk-rgrid-u-md-1-3 tk-stl-label">Index</p> <div class="tk-rgrid-u-1 tk-rgrid-u-md-2-3"> <p><span class="value">{{$parent.index}} </span></p> </div> </div> <search:field name="age" styling="label-left" label="Age"></search:field> <search:field name="date" styling="label-left" label="Created Date" date-format="dd/MM/yy"></search:field> <search:field name="blank" styling="label-left" label="Contact Number" default-value="No Contact Number on File."></search:field> <search:field name="number" styling="label-left" label="Worth" number-format="£#,###.##"></search:field> <search:field name="madeup" styling="label-left" label="Test"></search:field> <search:field name="longstring" styling="label-left" label="Long String" max-characters="40" show-more> </search:field> </div> </layout:block> </search:result> </search:result-list> </layout:grid> <layout:grid> <layout:block> </layout:grid>
```

```
angular.module('lightning') .controller('ExampleController', ['$scope','$timeout','ResponseService', function($scope,$timeout,ResponseService) { $scope.response = { page: 2, query: { rpp: 50 }, hits:{ act:['long','1234'] }, facets: { }, results: [ { result: { fields: { name: { name: 'name', val: ['Testing Tester'] }, url: { name: 'url', val: ['http://twigkit.com'] }, age: { name: 'age', val: ['Long', 35] }, date: { name: 'date', val: ['2014-11-11T00:00:00'] }, blank:{ name:'blank', val:null, }, number:{ name:'number', val:[123456789] }, longstring:{ name:'longstring', val:['Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.'] } }, }, }, { result: { fields: { name: { name: 'name', val: ['Foo (Bar)'] }, url: { name: 'url', val: ['http://twigkit.com'] }, age: { name: 'age', val: ['Long', 26] }, date: { name: 'date', val: ['2015-11-11T00:00:00'] }, blank:{ name:'blank', val:null, }, number:{ name:'number', val:[-123456789] } }, }, }, ], } $scope.results = $scope.response.results; ResponseService.setResponse('response', $scope.response); }]);
```

Result


Accessing the index of a result ($parent.index)
Testing Tester
Index 0
Age Long 35
Created Date 11/11/14
Contact Number No Contact Number on File.
Worth £123,456,789
Long String Lorem ipsum dolor sit amet, consectetur ...
show more
Foo (Bar)
Index 1
Age Long 26
Created Date 11/11/15
Contact Number No Contact Number on File.
Worth -£123,456,789

search:results-per-page

Description

The Results Per Page Directive allows you to change the number of documents that are returned from the platform. The results per page tag should be wrapped in a *Layout Controls Tag*.

Usage

as element:

```
<search:results-per-page response="{object}" [title="{string}"] [options="
{string}"]> </search:results-per-page>
```

Directive info

- This directive creates new scope.

Parameters

Param	Type	Details
response	**object**	A response object
title *(optional)*	**string**	The title to display before the result count options.
options *(optional)*	**string**	Which options to provide the user. Should be a comma-separated list of integers.

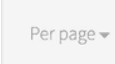
Example

Source

```
<layout:controls> <search:results-per-page response="response"
options="10,20,25,50"></search:results-per-page> </layout:controls>
```

```
angular.module('lightning') .controller('ExampleController', ['$scope',
function($scope) { $scope.response = { } }]);
```

Result



search:sort

Description

The sort directive for selecting how the data on the page should be sorted. The sort tag should be wrapped in a *Layout Controls Tag*.

Usage

as element:

```
<search:sort response="{string}" fields="{string}" [default-label="{string}"]
[title="{string}"] [default-order="{string}"] [mode="{string}"]> </search:sort>
```

Directive info

- This directive creates new scope.

Parameters

Param	Type	Details
response	**string**	The name of the response
fields	**string**	Fields to provide sorting on (comma-separated).
default-label *(optional)*	**string**	Label to display for the 'default' sort option when no specific sorts are applied. Usually implies sorting on score or relevance. Default: 'relevance'
title *(optional)*	**string**	Add a title to display. Default: 'Sort by'
default-order *(optional)*	**string**	Attribute to set the default sort order of Sort Fields (asc or desc). Default: 'asc'
mode *(optional)*	**string**	The mode in which the sorting should work, be default click. click: Selecting a sort will sort in the default order, when clicking on a selected sort this sort will be removed, to change the order use the ascending and descending options at the bottom of the dropdown. toggle: Selecting a sort will sort in the default order, when clicking on the selected sort again it will reverse the order.

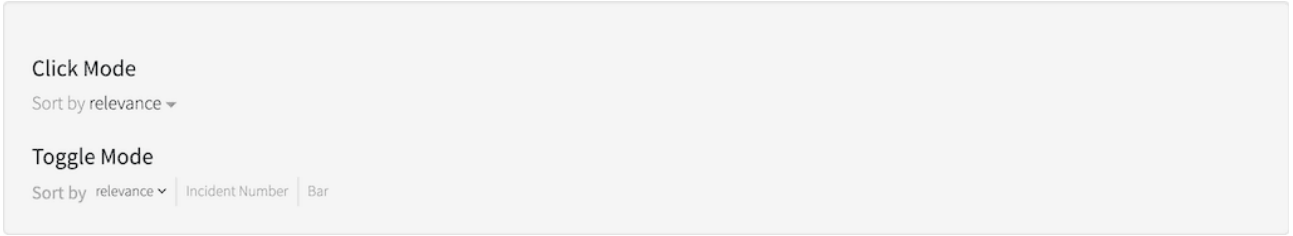
Example

Source

```
<layout:grid> <layout:block md="1-1"> <h5 id="example_source_click-mode">Click
Mode</h5> <layout:controls> <search:sort response="response"
fields="IncidentNum=Incident Number,Foo=Bar"></search:sort> </layout:controls>
</layout:block> <layout:grid> <layout:block md="1-1"> <h5
id="example_source_toggle-mode">Toggle Mode</h5> <layout:controls> <search:sort
response="response" fields="IncidentNum=Incident Number,Foo=Bar" mode="toggle"
class="left"></search:sort> </layout:controls> </layout:block> </layout:grid>
```

```
angular.module('lightning').controller('ExampleController', ['$scope',
'ResponseService', function($scope, ResponseService) { var response = {};
ResponseService.setResponse('response', response); }]);
```

Result



search:tabs

Description

The Tabs Directive

Usage

as element:

```
<search:tabs response="{object}" tabs="{string}" [field-name="{string}"] [all-label="{string}"] [platform="{string}"] [query="{string}"] [count-number="{string}"]> </search:tabs>
```

Directive info

- This directive creates new scope.

Parameters

Param	Type	Details
response	**object**	A response object
tabs	**string**	The tabs to output. Specified as a comma separated list. When using filter tabs, you can either specify the actual filter name, or the actual filter name with an alternative display name. To substitute actual filter names with alternative display names use the following notation: 'programme=Programmes'. . Multiple filters can be grouped under a single tab by separating the actual filter names with a |character, e.g.: 'Videos|Podcasts=Multimedia'.
field-name *(optional)*	**string**	The field to apply filters to when in 'filter' mode.
all-label *(optional)*	**string**	Label for the option to turn off all filters. If this attribute is specified, an additional link will be displayed to show all Results (e.g. not applying a Filter).
platform *(optional)*	**string**	The Platform to use when submitting the facet without any filters applied. This is used to to get the filter counts for the original query after a user clicks on a filter (e.g. drills down).
query *(optional)*	**string**	The name of the query object used to get the count values.
count-number *(optional)*	**string**	Specify how the number representing the count should be formatted. Optional values are 'plain', 'formatted', 'rounded' and 'none'.

Events

tab_changed

When the tab is changed on this directive an event called tab_changed will be fired with the name of the tab that is now active.

This event can be listened to like so.

```
$scope.$on('tab_changed', function(e, tab){
     console.log('New Tab = ' + tab);
});
```

Type:

broadcast

Target:

root scope

search:typeahead

Description

Creates a typeahead style search box that can be customized to allow the user to add filters using the *search keyword tag*. You can also include the *instant result list tag* to allow the user to select a result based on the search query and filters.

Usage

as element:

```
<search:typeahead platform="{string}" [auto-search="{string}"] [action="{string}"]>
</search:typeahead>
```

Parameters

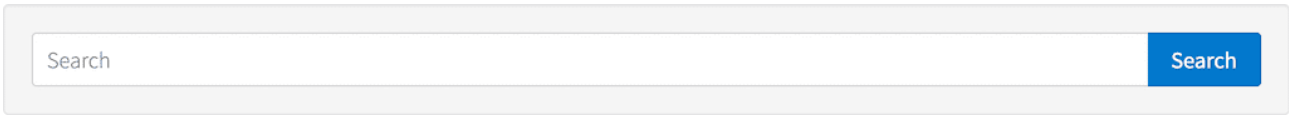
Param	Type	Details
platform	**string**	The name of the platform to use to search for matching keyword values
auto-search *(optional)*	**string**	Milliseconds to wait after the search is automatically submitted
action *(optional)*	**string**	The page to send the search query to

Example

Source

```
<search:typeahead platform="platform" datepicker-min-view="year" action="laureates"
class="searchbox"> <search:keyword keyword="country" facet="Country of affiliation"
description="Country of affiliation"></search:keyword> <search:keyword
keyword="gender" facet="gender" description="laureate's gender"></search:keyword>
<search:keyword keyword="category" facet="Prize Category" description="Award
category"></search:keyword> <search:keyword keyword="before" facet="year"
date="before" description="Awarded before this year" date-format="yyyy" date-
display-format="yyyy"></search:keyword> <search:keyword keyword="after"
facet="year" date="after" description="Awarded after this year" date-format="yyyy"
date-display-format="yyyy"></search:keyword> <search:keyword keyword="year"
facet="year" date="on" description="Awarded in this year" date-format="yyyy" date-
display-format="yyyy"></search:keyword> <instant:result-list platform="platform">
<search:result> <search:field name="full_name" label="" styling="label-inline"
url="/#/laureates/{{result.fields.id.val | actualValue}}"></search:field>
<search:field name="Prize Category" label="" styling="label-inline" class="pill">
</search:field> </search:result> </instant:result-list> </search:typeahead>
```

Result



security:user-actions

Description

User actions tag to log in, log out and navigate home.

To enable this you need to make sure that your security.conf file contains the line `webservice-enabled: true`

Your security.conf file can be found in `resources/conf/security/security.conf`.

For spring security you will need the following rule to allow access to the user api, if parts of the site are public.

```
<intercept-url pattern="/twigkit/api/user" access="isAnonymous() or isAuthenticated()"/>
```

Usage

as element:

```
<security:user-actions> </security:user-actions>
```

Directive info

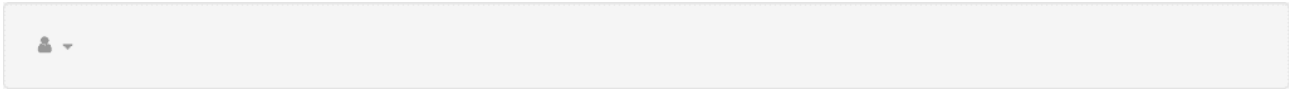
- This directive creates new scope.

Example

Source

```
<security:user-actions></security:user-actions>
```

Result



series:facet

Description

To render information in facets use the Series Facet tag, has to be nested within a *Chart Display tag*.

You can use the `response` attribute to load the data from the response or alternatively you can add a platform and this will get only the data the series needs to be rendered. You will also need to include the `query` directly on this tag if you wish for the user to be able to filter using the data of this series or alternatively add the `query` to the *Chart Display tag*.

For detail on how to change the color and background color see the demo below, for more detail on what you can change and set using CSS see here. *(http://www.highcharts.com/docs/chart-design-and-style/style-by-css)*

Usage

as element:

```
<series:facet response="{object}" platform="{string}" name="{string}" type="
{string}" [title="{string}"] [show-marker="{string}"] [number-to-show="{number}"]
[action="{string}"] [query="{object}"] [date-format="{string}"] [measure="
{string}"] [selected-by-default="{boolean}"] [select="{string}"]> </series:facet>
```

Directive info

- This directive creates new scope.

Parameters

Param	Type	Details
response	**object**	A response object (please include either a response or a platform not both!).
platform	**string**	The name of the platform to get the data from (please include either a response or a platform not both!).
name	**string**	Name of facet.
type	**string**	Chart type
title *(optional)*	**string**	The title of the series that appears in the legend and tooltip.
show-marker *(optional)*	**string**	Whether to show marker or not.
number-to-show *(optional)*	**number**	Number of filters to render on the graph.
action *(optional)*	**string**	The page/action to which clicking the chart should be submitted. By default the action points to the current page.
query *(optional)*	**object**	The search query from which to build links.
date-format *(optional)*	**string**	How to format the date. See #/api/lightning.filter:dateFormat[date format filter] for valid formats.
measure *(optional)*	**string**	What fact to use as an alternative to count as quantitative value for a filter.
selected-by-default *(optional)*	**boolean**	Whether this series should be shown when the chart:display is using select-series="true" mode. Default: true.

Param	Type	Details
select *(optional)*	**string**	How the data should be filtered when clicking on a part of the chart, a valid query must be present. Valid values: • **false** User selects a filter by clicking the value; **(default)** • **multiselect** Supports selecting multiple filters that get combined with an 'OR'. • **multiselect-instant** Supports selecting multiple filters that get combined with an 'OR', filters will be applied automatically.

Example

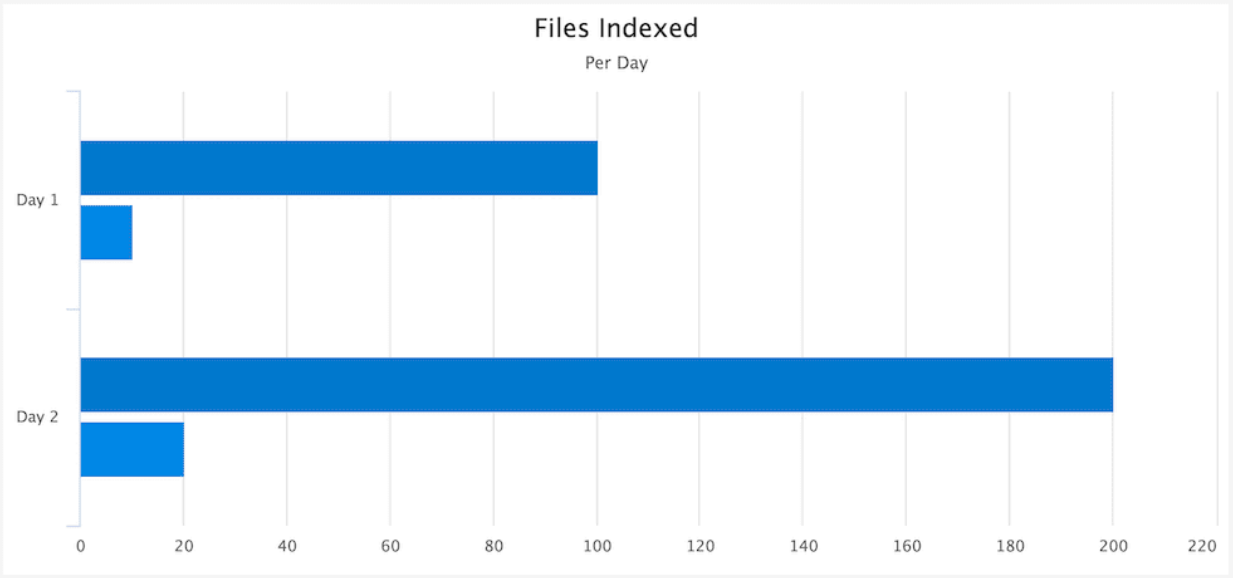
Source

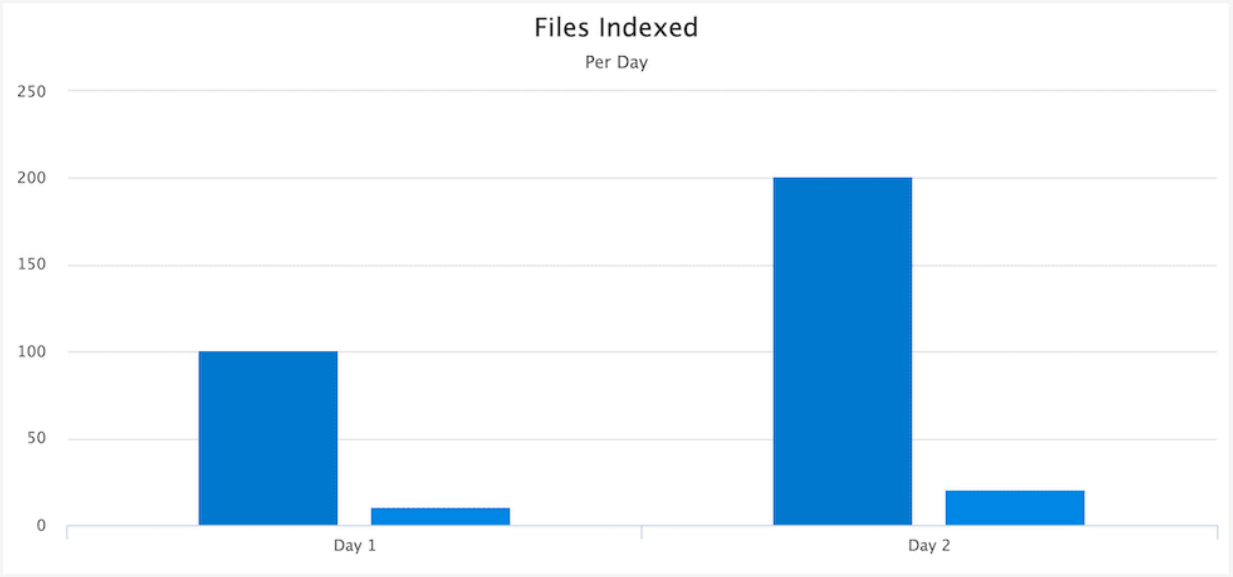
```
<chart:display response="response" title="Files Indexed" subtitle="Per Day"
type="bar"> <series:facet name="test1" title="First Pass"></series:facet>
<series:facet name="test2" title="Second Pass"></series:facet> </chart:display> <hr
/> <chart:display response="response" title="Files Indexed" subtitle="Per Day">
<series:facet name="test1" type="column" title="First Pass"></series:facet>
<series:facet name="test2" type="column" title="Second Pass"></series:facet>
</chart:display> <hr /> <div> <h3 id="example_number-to-show-attribute">Number To
Show Attribute</h3> <chart:display response="response" title="Population"
type="line"> <series:facet name="population" title="Current Population" number-to-
show="5"></series:facet> <series:facet name="population2000" title="Population in
2000" number-to-show="5"></series:facet> </chart:display> </div> <div> <h3
id="example_show-marker-attribute">Show Marker Attribute</h3> <chart:display
response="response" title="Population" type="line"> <series:facet name="population"
title="Current Population" show-marker="false"></series:facet> <series:facet
name="population2000" title="Population in 2000" show-marker="false">
</series:facet> </chart:display> </div> <div id="colour-demo"> <h3
id="example_colour">Colour (since 0.3.0)</h3> <p>Add an ID to the element wrapping
your chart and configure the background colour and series colours like so.</p>
<pre> #colour-demo .highcharts-color-0{ fill:pink; stroke:pink; } #colour-demo
.highcharts-color-1{ fill:yellow; stroke:yellow; } #colour-demo .highcharts-
background{ fill:#f5f0f0; } </pre> <p>For more detail on what you can change and
set using CSS see <a href="http://www.highcharts.com/docs/chart-design-and-
style/style-by-css" target="_blank">here.</a></p> <chart:display
response="response" title="Population" type="column"> <series:facet
name="population" title="Current Population"></series:facet> <series:facet
name="population2000" title="Population in 2000"></series:facet> </chart:display>
```

```
#colour-demo .highcharts-color-0 { fill: pink; stroke: pink; colour-demo
.highcharts-color-1 { fill: yellow; stroke: yellow; colour-demo .highcharts-
background { fill: #f5f0f0;
```

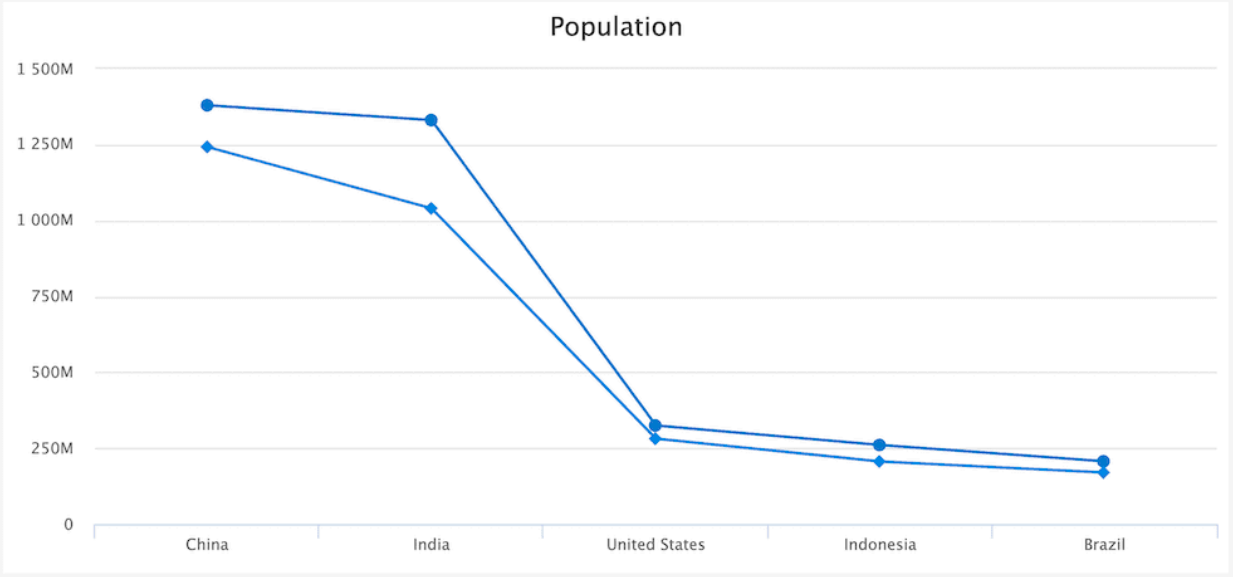
```
angular.module('lightning') .controller('ExampleController', ['$scope','$timeout','ResponseService', function($scope,$timeout, ResponseService) { $scope.response = { page: 2, query: { rpp: 50 }, facets: { test1: { filters: [ { val: { dsp: 'Day 1', act: 'Day 1', }, count: 100, }, { val: { dsp: 'Day 2', act: 'Day 2', }, count: 200, } ] }, test2: { filters: [ { val: { dsp: 'Day 1', act: 'Day 1', }, count: 10, }, { val: { dsp: 'Day 2', act: 'Day 2', }, count: 20, } ] }, population:{ filters: [ { val: 'China', count: 1379790000, }, { val: 'India', count: 1330780000, }, { val: 'United States', count: 324897000, }, { val: 'Indonesia', count: 260581000, }, { val: 'Brazil', count: 206927000, }, { val: 'Pakistan', count: 194772000, }, { val: 'Nigeria', count: 186987000, }, { val: 'Bangladesh', count: 161401000, }, { val: 'Russia', count: 146727405, }, { val: 'Japan', count: 126693000, }, ] }, population2000:{ filters: [ { val: 'China', count: 1242612300, }, { val: 'India', count: 1040000000, }, { val: 'United States', count: 281421923, }, { val: 'Indonesia', count: 206264595, }, { val: 'Brazil', count: 170000000, }, { val: 'Pakistan', count: 140000000, }, { val: 'Nigeria', count: 110000000, }, { val: 'Bangladesh', count: 130000000, }, { val: 'Russia', count: 147000000, }, { val: 'Japan', count: 127000000, }, ] }, }, results: [] } ResponseService.setResponse('response', $scope.response); }]);
```

Result

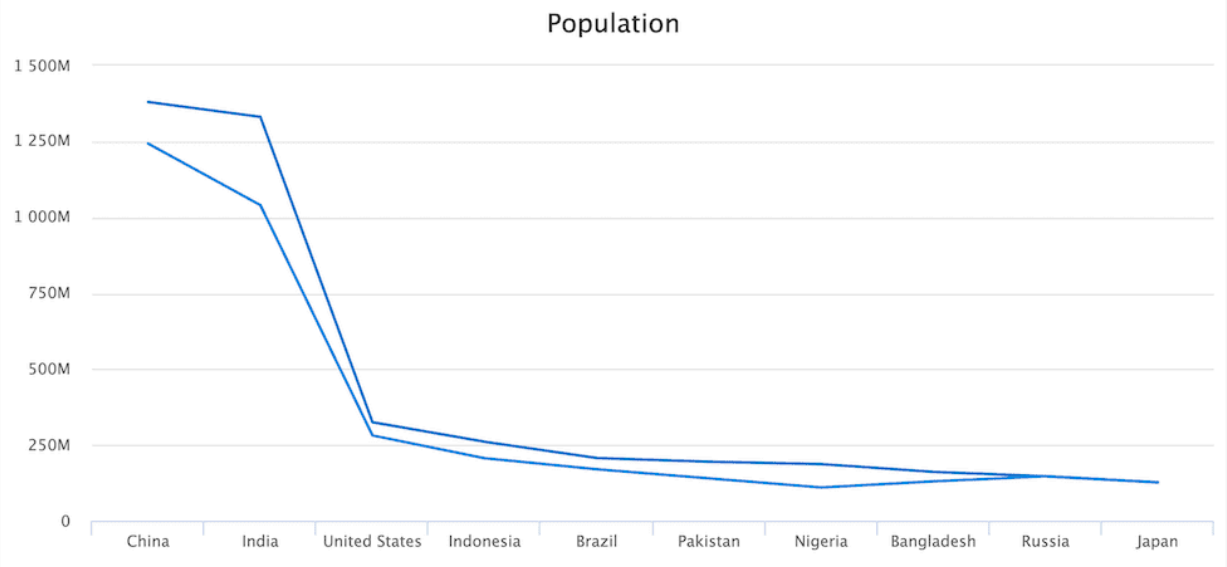




Number To Show Attribute



Show Marker Attribute

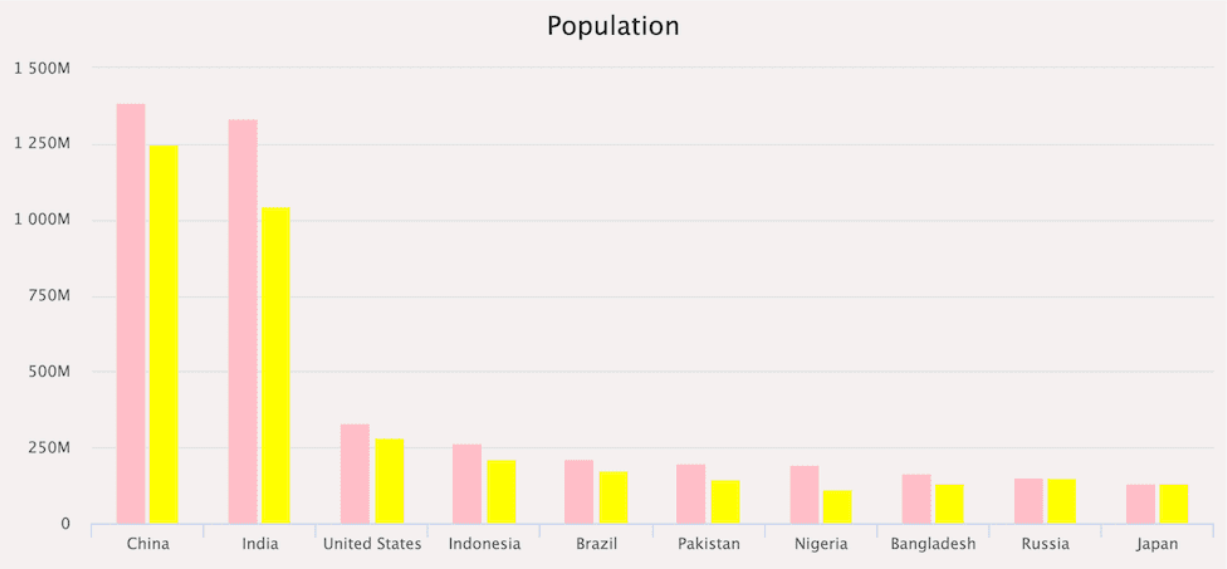


Colour (since 0.3.0)

Add an ID to the element wrapping your chart and configure the background colour and series colours like so.

```
#colour-demo .highcharts-color-0{
  fill:pink;
  stroke:pink;
}
#colour-demo .highcharts-color-1{
  fill:yellow;
  stroke:yellow;
}
#colour-demo .highcharts-background{
  fill:#f5f0f0;
}
```

For more detail on what you can change and set using CSS see here.



series:pivot

Description

To render information in a pivot facet use the Series Pivot tag, this tag has to be nested within a *Chart Display tag*. Please note it will only render the second level of relevant filters. For more details see the data object used in the examples below.

You can use the `response` attribute to load the data from the response. You will also need to include the `query` directly on this tag if you wish for the user to be able to filter using the data of this series or alternatively add the `query` to the *Chart Display tag*.

For detail on how to change the color and background color see the demo below, for more detail on what you can change and set using CSS see here. *(http://www.highcharts.com/docs/chart-design-and-style/style-by-css)*

When rendering pivot facet data it is recommended that you always include the *Chart Legend tag* within your *Chart Display tag* as the examples below show.

Usage

as element:

```
<series:pivot response="{string}" name="{string}" type="{string}" [show-marker="
{string}"] [action="{string}"] [query="{object}"] [select="{string}"]>
</series:pivot>
```

Directive info

- This directive creates new scope.

Parameters

Param	Type	Details
response	**string**	The name of the response to get the facet from.
name	**string**	Name of facet.
type	**string**	Chart type
show-marker *(optional)*	**string**	Whether to show marker or not.
action *(optional)*	**string**	The page/action to which clicking the chart should be submitted. By default the action points to the current page.
query *(optional)*	**object**	The search query from which to build links.
select *(optional)*	**string**	How the data should be filtered when clicking on a part of the chart, a valid query must be present. Valid values: • **false** User selects a filter by clicking the value; **(default)** • **multiselect** Supports selecting multiple filters that get combined with an 'OR'. • **multiselect-instant** Supports selecting multiple filters that get combined with an 'OR', filters will be applied automatically.

series:result-list

Description

To render information using fields from a result list you can use the Series Result List tag and use the label-field and value-field attributes. This tag must be nested in the *Chart Display tag*.

You can use the `response` attribute to load the data from the response or alternatively you can add a platform and this will get only the data the series needs to be rendered. You will also need to include the `query` directly on this tag if you wish for the user to be able to filter using the data of this series or alternatively add the `query` to the *Chart Display tag*.

For detail on how to change the color and background color see the demo below, for more detail on what you can change and set using CSS see here. *(http://www.highcharts.com/docs/chart-design-and-style/style-by-css)*

Usage

as element:

```
<series:result-list response="{object}" results="{object}" platform="{string}"
label-field="{string|expression}" value-field="{string|expression}" [class-field="
{string|expression}"] type="{string}" [title="{string|expression}"] [show-marker="
{string}"] [url-field="{string}"] [number-to-show="{number}"] [action="{string}"]
[query="{object}"] [selected-by-default="{boolean}"] [select="{string}"]>
</series:result-list>
```

Directive info

- This directive creates new scope.

Parameters

Param	Type	Details
response	**object**	A response object (please include either a response, results, or a platform not multiple!).
results	**object**	An array of results (please include either a response, results, or a platform not multiple!).
platform	**string**	The name of the platform to get the data from (please include either a response, results, or a platform not multiple!).
label-field	**string]link: [expression**	Name of field containing the label.
value-field	**string]link: [expression**	Name of field containing the value.
class-field *(optional)*	**string]link: [expression**	Name of the field containing the value you wish to add as a class to the point
type	**string**	The type of chart to render. Types available - 'pie', 'line', 'area', 'column', 'bar', or 'scatter'.
title *(optional)*	**string]link: [expression**	The title of the series that appears in the legend and tooltip.
show-marker *(optional)*	**string**	Whether to show marker or not.
url-field *(optional)*	**string**	The field in the result list containing the URL from which to make the visualisation clickable.
number-to-show *(optional)*	**number**	Number of results to render on the graph.
action *(optional)*	**string**	The page/action to which clicking the chart should be submitted. By default the action points to the current page.
query *(optional)*	**object**	The search query from which to build links.
selected-by-default *(optional)*	**boolean**	Whether this series should be shown when the chart:display is using select-series="true" mode. Default: true.

Param	Type	Details
select *(optional)*	**string**	How the data should be filtered when clicking on a part of the chart, a valid query must be present. Valid values: • **false** User selects a filter by clicking the value; **(default)** • **multiselect** Supports selecting multiple filters that get combined with an 'OR'. • **multiselect-instant** Supports selecting multiple filters that get combined with an 'OR', filters will be applied automatically.

Example

Source

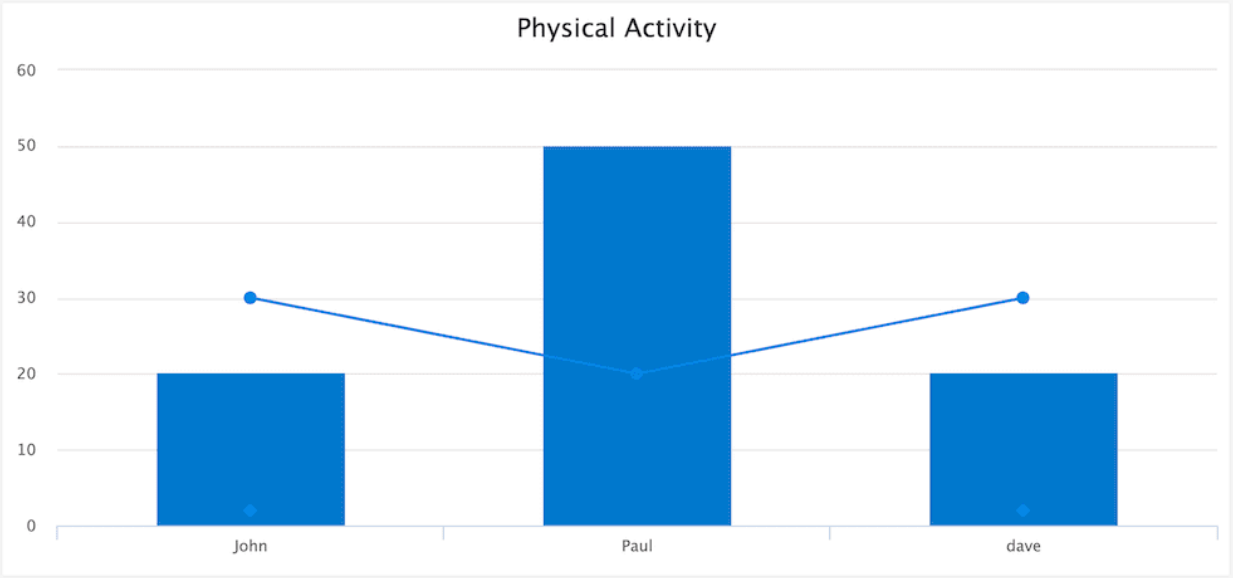
```
<h3 id="example_tooltip-tag">Tooltip Tag</h3> <chart:display response="response"
title="Physical Activity"> <series:result-list response="response" label-
field="name" value-field="runs" type="column"></series:result-list> <series:result-
list response="response" label-field="name" value-field="rides" type="line">
</series:result-list> <series:result-list response="response" label-field="name"
value-field="swims" type="area"></series:result-list> </chart:display> <div
id="colour-demo"> <h3 id="example_colour">Colour (since 0.3.0)</h3> <p>Add an ID to
the element wrapping your chart and configure the background colour and series
colours like so.</p> <pre> #colour-demo .highcharts-color-0{ fill:pink;
stroke:pink; } #colour-demo .highcharts-color-1{ fill:yellow; stroke:yellow; }
#colour-demo .highcharts-color-2{ fill:purple; stroke:purple; } #colour-demo
.highcharts-background{ fill:#f5f0f0; } </pre> <p>For more detail on what you can
change and set using CSS see <a href="http://www.highcharts.com/docs/chart-design-
and-style/style-by-css" target="_blank">here.</a></p> <chart:display
response="response" title="Physical Activity"> <series:result-list
response="response" label-field="name" value-field="runs" type="column">
</series:result-list> <series:result-list response="response" label-field="name"
value-field="rides" type="line"></series:result-list> <series:result-list
response="response" label-field="name" value-field="swims" type="area">
</series:result-list> </chart:display> </div> <div id="class-field"> <h3
id="example_class-field">Class Field</h3> <p>Add an ID to the element wrapping your
chart and configure the point based on the content of the class field you provide.
</p> <pre> #class-field .athlete{ fill:teal; stroke:teal; } #class-field .iron-man{
fill:red; stroke:red; } #class-field .cyclist{ fill:blue; stroke:blue; } </pre>
<p>For more detail on what you can change and set using CSS see <a
href="http://www.highcharts.com/docs/chart-design-and-style/style-by-css"
target="_blank">here.</a></p> <chart:display response="response" title="Physical
Activity"> <series:result-list response="response" label-field="name" value-
field="runs" type="column" class-field="type"></series:result-list>
</chart:display>
```

```
#colour-demo .highcharts-color-0 { fill: pink; stroke: pink; colour-demo
.highcharts-color-1 { fill: yellow; stroke: yellow; colour-demo .highcharts-color-2
{ fill: purple; stroke: purple; colour-demo .highcharts-background { fill: #f5f0f0;
class-field .athlete { fill: teal; stroke: teal; class-field .iron-man { fill: red;
stroke: red; class-field .cyclist { fill: blue; stroke: blue;
```

```
angular.module('lightning') .controller('ExampleController',
['$scope','$timeout','ResponseService', function($scope,$timeout,ResponseService) {
$scope.response = { page: 2, query: { rpp: 50 }, facets: { }, results: [ { result:
{ fields: { name:{ val:'John', }, runs: { val:20, }, rides:{ val:30, }, swims:{
val:2, }, type:{ val:'cyclist' } } }, }, { result: { fields: { name: { val:'Paul',
}, runs: { val:50, }, rides:{ val:20, }, type:{ val:'athlete' } } }, hits: { act:
['Hits', 264] } }, { result: { fields: { name:{ val:'dave', }, runs: { val:20, },
rides:{ val:30, }, swims:{ val:2, }, type:{ val:'iron man' } } }, }, ] }
ResponseService.setResponse('response',$scope.response); }]);
```

Result

Tooltip Tag

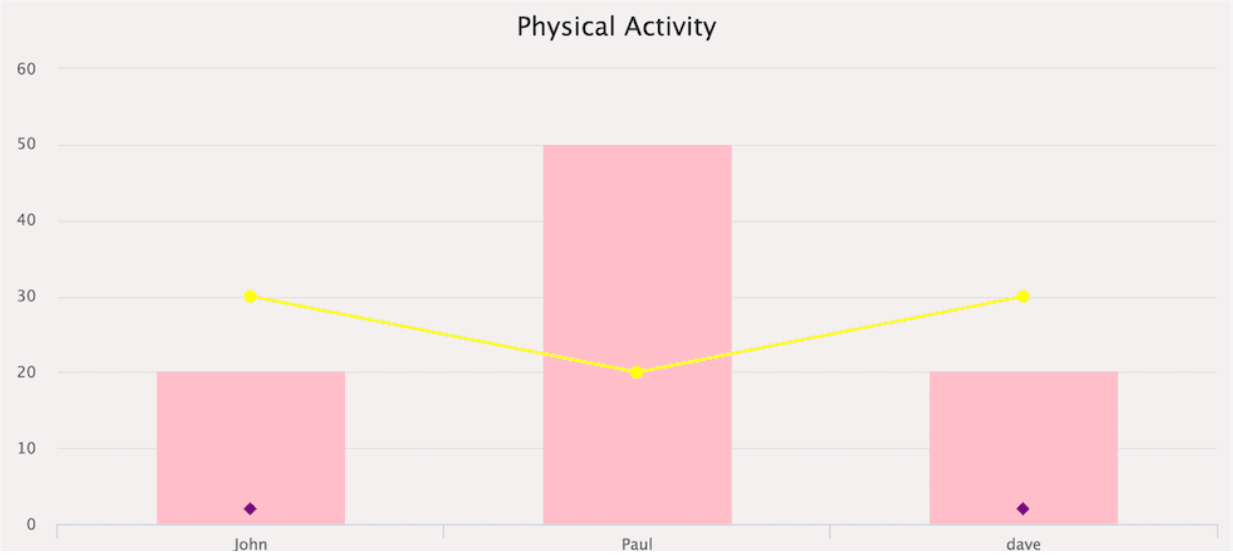


Colour (since 0.3.0)

Add an ID to the element wrapping your chart and configure the background colour and series colours like so.

```
#colour-demo .highcharts-color-0{
  fill:pink;
  stroke:pink;
}
#colour-demo .highcharts-color-1{
  fill:yellow;
  stroke:yellow;
}
#colour-demo .highcharts-color-2{
  fill:purple;
  stroke:purple;
}
#colour-demo .highcharts-background{
  fill:#f5f0f0;
}
```

For more detail on what you can change and set using CSS see here.

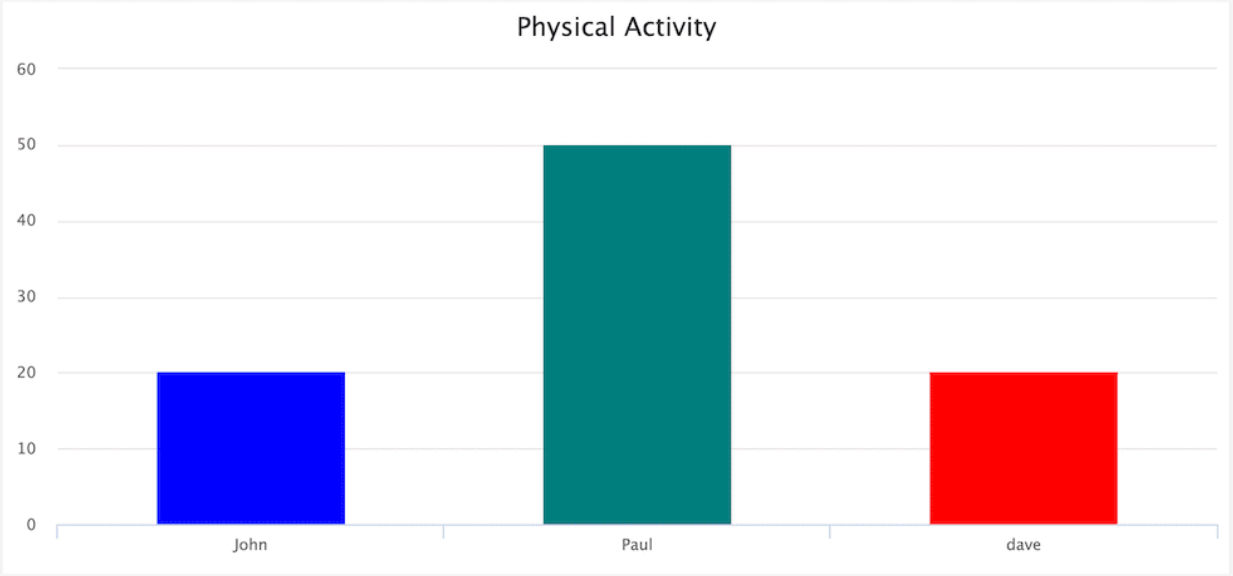


Class Field

Add an ID to the element wrapping your chart and configure the point based on the content of the class field you provide.

```
#class-field .athlete{
  fill:teal;
  stroke:teal;
}
#class-field .iron-man{
  fill:red;
  stroke:red;
}
#class-field .cyclist{
  fill:blue;
  stroke:blue;
}
```

For more detail on what you can change and set using CSS see here.



series:time

Description

The series time directive shows a time series on a chart, in order to use this you must set type to 'time' on the *chart display tag*.

For detail on how to change the color and background color see the demo below, for more detail on what you can change and set using CSS see here. *(http://www.highcharts.com/docs/chart-design-and-style/style-by-css)*

Usage

as element:

```
<series:time response="{object}" platform="{string}" name="{string}" value-field="
{string|expression}" label-field="{string|expression}" [title="
{string|expression}"] [show-marker="{string}"] [type="{string}"] [range-selector="
{boolean}"] [navigator="{boolean}"] [query="{object}"] [measure="{string}"] [sort="
{boolean}"] [number-to-show="{Number}"] [selected-by-default="{boolean}"]>
</series:time>
```

Directive info

- This directive creates new scope.

Parameters

Param	Type	Details
response	**object**	A response object (please include either a response or a platform not both!).
platform	**string**	The name of the platform to get the data from (please include either a response or a platform not both!).
name	**string**	Name of facet. (Use either name or value-field and label-field not both!)
value-field	**string]link: [expression**	Field from the result containing the count. (Use either name or value-field and label-field not both!)
label-field	**string]link: [expression**	Field from the result containing the label. (Use either name or value-field and label-field not both!)
title *(optional)*	**string]link: [expression**	The title of the series that appears in the legend and tooltip.
show-marker *(optional)*	**string**	Whether to show marker or not.
type *(optional)*	**string**	Type of chart to render.
range-selector *(optional)*	**boolean**	Where the input boxes should be enabled (from and to) Default:true
navigator *(optional)*	**boolean**	Whether to enable the navigator beneth the chart. Default:true
query *(optional)*	**object**	The search query from which to build links.
measure *(optional)*	**string**	What fact to use as an alternative to count as quantitative value for a filter.
sort *(optional)*	**boolean**	Whether the data should be sort by the directive. Defaults to true.
number-to-show *(optional)*	**Number**	The number of points to show on the graph. Applies only when using the platform attribute. (Default:25)
selected-by-default *(optional)*	**boolean**	Whether this series should be shown when the chart:display is using select-series="true" mode. Default: true.

show:when

Description

Show or hide element based on expression passed to it.

Usage

as attribute

```
<ANY show-when="{expression}"> ... </ANY>
```

Parameters

Param	Type	Details
show-when	**expression**	Expression to evaluate.

Example

Source

```
<p><a href="javascript:void(0);" ng-click="alert()" styling="button">{{foo ? 'Hide' : 'Show'}}!</a> </p> <h3 show-when="foo" id="example_hello-world">Hello World</h3>
```

```
angular.module('lightning') .controller('ExampleController', ['$scope','$timeout', function($scope,$timeout) { $scope.foo = false; $scope.alert = function(){ $scope.foo = !$scope.foo }; }]);
```

Result

slide:toggle

Description

The toggle panes directive

Usage

as attribute

```
<ANY slide-toggle target="{string}"> ... </ANY>
```

Directive info

- This directive creates new scope.

Parameters

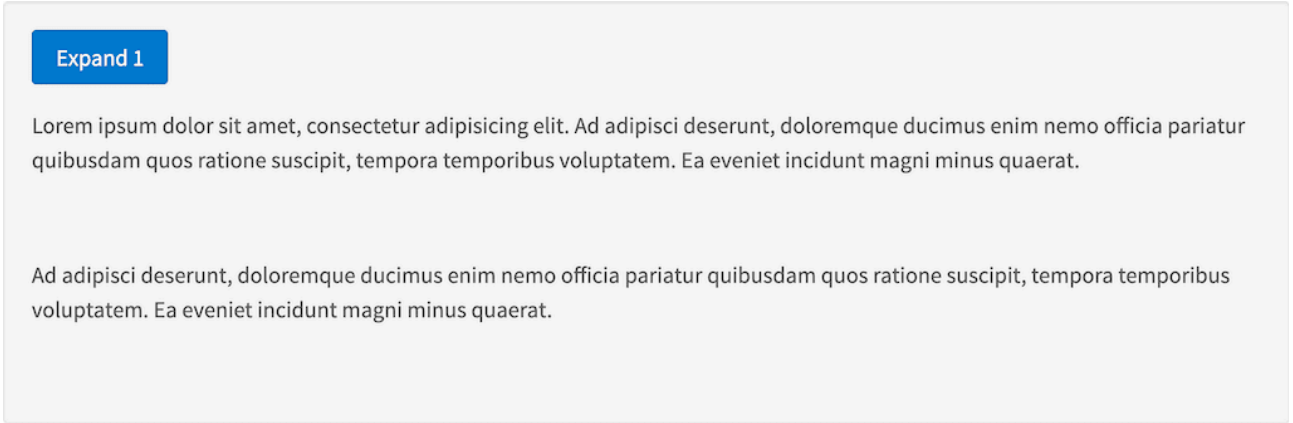
Param	Type	Details
target	**string**	CSS selector to get slideable element

Example

Source

```
<button class="tk-stl-button" widget-slide-toggle="#foo1">Expand 1</button> <div id="foo1" class="widget-slide-down"> <p>Lorem ipsum dolor sit amet, consectetur adipisicing elit. Ad adipisci deserunt, doloremque ducimus enim nemo officia pariatur quibusdam quos ratione suscipit, tempora temporibus voluptatem. Ea eveniet incidunt magni minus quaerat.</p> <br> <p padding-bottom="2">Ad adipisci deserunt, doloremque ducimus enim nemo officia pariatur quibusdam quos ratione suscipit, tempora temporibus voluptatem. Ea eveniet incidunt magni minus quaerat.</p> </div>
```

Result



social:comment-form

Deprecated API

Description

Social Comment Form directive has been deprecated, please see *Collaborate Comment Form tag*

Usage

as element:

```
<social:comment-form> </social:comment-form>
```

social:comment-list

Deprecated API

Description

Social Comment List directive has been deprecated, please see *Collaborate Comment List tag*

Usage

as element:

```
<social:comment-list> </social:comment-list>
```

styling

Description

Extends the HTML style attribute to apply twigkit classes to the element as specified in the style attribute

Usage

as attribute

```
<ANY styling="{string}"> ... </ANY>
```

Parameters

Param	Type	Details
styling	**string**	Space separated list of styles to apply to the element

table:body

Description

Creates a table body `<tbody>`

Usage

as element:

```
<table:body [infinite-scrolling="{boolean}"] [query="{string}"] [platform="
{object}"]> </table:body>
```

Parameters

Param	Type	Details
infinite-scrolling *(optional)*	**boolean**	Whether infiniteScrolling should be enabled. (Not supported when using results instead of response in table:result-list)
query *(optional)*	**string**	The name of the query used for infinite scrolling or load more. (Not supported when using results instead of response in table:result-list)
platform *(optional)*	**object**	The platform object used for infinite scrolling or load more. (Not supported when using results instead of response in table:result-list)

table:cell

Description

Creates a table cell `<td>`

Usage

as element:

```
<table:cell name="{string}" [default-value="{string}"] [separator="{string}"]
[date-format="{string}"] [number-format="{string}"] [show="{Number}"] [prefix="
{string}"] [suffix="{string}"] [url="{string}"] [url-field="{string}"] [target="
{string}"] [action="{string}"] [query="{string}"]> </table:cell>
```

Directive info

- This directive creates new scope.

Parameters

Param	Type	Details
name	**string**	Name of the field to render
default-value *(optional)*	**string**	Default value to show when no value is present in the index.
separator *(optional)*	**string**	Separator character or sequence to split a Field into multiple values. Setting the separator to “;” would turn the value “company1; company2; company3” into a list of 3 values.
date-format *(optional)*	**string**	How to format the date. See #/api/lightning.filter:dateFormat[date format filter] for valid formats.
number-format *(optional)*	**string**	Format string for the numbers used using the Java DecimalFormat pattern. Default: ',0.'
show *(optional)*	**Number**	Number of values to show if the field is multivalued. Default: No limit (-1)
prefix *(optional)*	**string**	Text to prepend to index values (not applied to default value). E.g. if the index field 'money' contains the value '10'. A prefix="£" would result in '£10' being displayed as the value.
suffix *(optional)*	**string**	Text to append to index values (not applied to default value). E.g. if the index field 'weight' contains the value '10'. A suffix=" kg." would result in '10 kg.' being displayed as the value.
url *(optional)*	**string**	URL to redirect to when the field is clicked;
url-field *(optional)*	**string**	Field containing a URL to redirect to when the field is clicked;
target *(optional)*	**string**	Set an HTML target attribute for the anchor tag. For example use _blank to have the generated link open the results in a new window. (Not applicable to query)

Param	Type	Details
action *(optional)*	**string**	Alternative page to submit the query parameters to - defaults to current page. If a 'url' attribute is provided the action is ignored and query parameters are not included in the request.
query *(optional)*	**string**	When set, the value of the named field will be displayed as a hyperlink that submits the query again with this field and its value as a search constraint.

table:header

Description

Creates a table header `<thead>`

Usage

as element:

```
<table:header> </table:header>
```

table:header-cell

Description

Creates a table header cell `<th>`

Usage

as element:

```
<table:header-cell [label="{string}"] [name="{string}"] [sort="{boolean}"]>
</table:header-cell>
```

Parameters

Param	Type	Details
label *(optional)*	**string**	Label for the header.
name *(optional)*	**string**	Name of the field to sort.
sort *(optional)*	**boolean**	Whether to enable sorting.

table:result-list

Description

Creates a table

The following tags can be nested within the table result list tag.

- *Table Header*
- *Table Header Cell*
- *Table Body*
- *Table Row*
- *Table Cell*

Usage

as element:

```
<table:result-list response="{string}" results="{array}"> </table:result-list>
```

Parameters

Param	Type	Details
response	**string**	The name of the response to populate the table with. (Use results or response not both.)
results	**array**	A list of results to be rendered in the table. (Use results or response not both.)

Example

Source

```
<h3 id="example_table">Table</h3> <table:result-list response="response">
<table:header> <table:header-cell sort="true">Name</table:header-cell>
<table:header-cell label="Urk"></table:header-cell> <table:header-cell name="Date">
</table:header-cell> <table:header-cell name="Number"></table:header-cell>
<table:header-cell name="multi"></table:header-cell> <table:header-cell
name="Prefix"></table:header-cell> <table:header-cell name="Suffix"></table:header-
cell> <table:header-cell name="Empty"></table:header-cell> </table:header>
<table:body> <table:row> <table:cell name="name"></table:cell> <table:cell
name="url" url-field="url" target="_blank"></table:cell> <table:cell name="date"
date-format="fullDate"></table:cell> <table:cell name="number" number-
format="$#,###.00"></table:cell> <table:cell name="multi" show="2"></table:cell>
<table:cell name="prefix" prefix="£"></table:cell> <table:cell name="suffix"
suffix=" kg." url="https://en.wikipedia.org/wiki/Kilogram"></table:cell>
<table:cell name="empty" default-value="No Value"></table:cell> </table:row>
</table:body> </table:result-list> <hr /> <h3 id="example_table">Table (with
results)</h3> <table:result-list results="response.results"> <table:header>
<table:header-cell sort="true">Name</table:header-cell> <table:header-cell
label="Urk"></table:header-cell> <table:header-cell name="Date"></table:header-
cell> <table:header-cell name="Number"></table:header-cell> <table:header-cell
name="multi"></table:header-cell> <table:header-cell name="Prefix"></table:header-
cell> <table:header-cell name="Suffix"></table:header-cell> <table:header-cell
name="Empty"></table:header-cell> </table:header> <table:body> <table:row>
<table:cell name="name"></table:cell> <table:cell name="url" url-field="url"
target="_blank"></table:cell> <table:cell name="date" date-format="fullDate">
</table:cell> <table:cell name="number" number-format="$#,###.00"></table:cell>
<table:cell name="multi" show="2"></table:cell> <table:cell name="prefix"
prefix="£"></table:cell> <table:cell name="suffix" suffix=" kg."
url="https://en.wikipedia.org/wiki/Kilogram"></table:cell> <table:cell name="empty"
default-value="No Value"></table:cell> </table:row> </table:body> </table:result-
list> <hr /> <h3 id="example_vertical-comparison">Vertical Comparison (class="tbl-
compare")</h3> <table:result-list response="response" class="tbl-compare">
<table:header> <table:header-cell sort="true">Name</table:header-cell>
<table:header-cell label="Urk"></table:header-cell> <table:header-cell name="Date">
</table:header-cell> <table:header-cell name="Number"></table:header-cell>
<table:header-cell name="multi"></table:header-cell> <table:header-cell
name="Prefix"></table:header-cell> <table:header-cell name="Suffix"></table:header-
cell> <table:header-cell name="Empty"></table:header-cell> </table:header>
<table:body> <table:row> <table:cell name="name"></table:cell> <table:cell
name="url" url-field="url" target="_blank"></table:cell> <table:cell name="date"
date-format="fullDate"></table:cell> <table:cell name="number" number-
format="$#,###.00"></table:cell> <table:cell name="multi" show="2"></table:cell>
<table:cell name="prefix" prefix="£"></table:cell> <table:cell name="suffix"
suffix=" kg." url="https://en.wikipedia.org/wiki/Kilogram"></table:cell>
<table:cell name="empty" default-value="No Value"></table:cell> </table:row>
</table:body> </table:result-list> <hr /> <h3 id="example_stacked">Stacked
(class="tbl-stacked")</h3> <table:result-list response="response" class="tbl-
stacked"> <table:header> <table:header-cell sort="true">Name</table:header-cell>
<table:header-cell label="Urk"></table:header-cell> <table:header-cell name="Date">
</table:header-cell> <table:header-cell name="Number"></table:header-cell>
<table:header-cell name="multi"></table:header-cell> <table:header-cell
name="Prefix"></table:header-cell> <table:header-cell name="Suffix"></table:header-
cell> <table:header-cell name="Empty"></table:header-cell> </table:header>
<table:body> <table:row> <table:cell name="name"></table:cell> <table:cell
name="url" url-field="url" target="_blank"></table:cell> <table:cell name="date"
date-format="fullDate"></table:cell> <table:cell name="number" number-
format="$#,###.00"></table:cell> <table:cell name="multi" show="2"></table:cell>
<table:cell name="prefix" prefix="£"></table:cell> <table:cell name="suffix"
suffix=" kg." url="https://en.wikipedia.org/wiki/Kilogram"></table:cell>
<table:cell name="empty" default-value="No Value"></table:cell> </table:row>
</table:body> </table:result-list> <hr /> <h3 id="example_compare-
combination">Compare Combination (class="tbl-combo-compare")</h3> <table:result-
```

```
list response="response" class="tbl-combo-compare"> <table:header> <table:header-cell sort="true">Name</table:header-cell> <table:header-cell label="Urk"> </table:header-cell> <table:header-cell name="Date"></table:header-cell> <table:header-cell name="Number"></table:header-cell> <table:header-cell name="multi"></table:header-cell> <table:header-cell name="Prefix"></table:header-cell> <table:header-cell name="Suffix"></table:header-cell> <table:header-cell name="Empty"></table:header-cell> </table:header> <table:body> <table:row> <table:cell name="name"></table:cell> <table:cell name="url" url-field="url" target="_blank"></table:cell> <table:cell name="date" date-format="fullDate"> </table:cell> <table:cell name="number" number-format="$#,###.00"></table:cell> <table:cell name="multi" show="2"></table:cell> <table:cell name="prefix" prefix="£"></table:cell> <table:cell name="suffix" suffix=" kg." url="https://en.wikipedia.org/wiki/Kilogram"></table:cell> <table:cell name="empty" default-value="No Value"></table:cell> </table:row> </table:body> </table:result-list>
```

```
angular.module('lightning') .controller('ExampleController', ['$scope','$timeout','ResponseService', function($scope,$timeout,ResponseService) { $scope.response = { page: 2, query: { rpp: 50 }, hits:{ act:['long','1234'] }, facets: { }, results: [ { result: { fields: { name: { name: 'name', val: ['Testing Tester'] }, url: { name: 'url', val: ['http://twigkit.com'] }, age: { name: 'age', val: ['Long', 35] }, date: { name: 'date', val: ['2014-11-11T00:00:00'] }, blank:{ name:'blank', val:null, }, number:{ name:'number', val:[123456789] }, multi:{ name:'multi', val:['Multi','Value','Field'] }, prefix:{ name:'prefix', val:['10'] }, suffix:{ name:'suffix', val:['10'] }, empty:{ name:'empty', val:[] } }, }, }, { result: { fields: { name: { name: 'name', val: ['Foo Bar'] }, url: { name: 'url', val: ['http://twigkit.com'] }, age: { name: 'age', val: ['Long', 26] }, date: { name: 'date', val: ['2015-11-11T00:00:00'] }, blank:{ name:'blank', val:null, }, number:{ name:'number', val:[-123456789] }, multi:{ name:'multi', val: ['Multi','Value','Field'] }, prefix:{ name:'prefix', val:['-10'] }, suffix:{ name:'suffix', val:['-10'] }, empty:{ name:'empty', val:null } }, }, }, ], } ResponseService.setResponse('response', $scope.response); }]);
```

Result

Table

Name	Urk	Date	Number	multi	Prefix	Suffix	Empty
Testing Tester	http://twigkit.com	Tuesday, November 11, 2014	$123,456,789.00	Multi Value	£10	10 kg.	No Value
Foo Bar	http://twigkit.com	Wednesday, November 11, 2015	-$123,456,789.00	Multi Value	£-10	-10 kg.	No Value

Table (with results)

Name	Urk	Date	Number	multi	Prefix	Suffix	Empty
Testing Tester	http://twigkit.com	Tuesday, November 11, 2014	$123,456,789.00	Multi Value	£10	10 kg.	No Value
Foo Bar	http://twigkit.com	Wednesday, November 11, 2015	-$123,456,789.00	Multi Value	£-10	-10 kg.	No Value

Vertical Comparison (class="tbl-compare")

Name	Urk	Date	Number	multi	Prefix	Suffix	Empty
Testing Tester	http://twigkit.com	Tuesday, November 11, 2014	$123,456,789.00	Multi Value	£10	10 kg.	No Value
Foo Bar	http://twigkit.com	Wednesday, November 11, 2015	-$123,456,789.00	Multi Value	£-10	-10 kg.	No Value

Stacked (class="tbl-stacked")

Name	Urk	Date	Number	multi	Prefix	Suffix	Empty
Testing Tester	http://twigkit.com	Tuesday, November 11, 2014	$123,456,789.00	Multi Value	£10	10 kg.	No Value
Foo Bar	http://twigkit.com	Wednesday, November 11, 2015	-$123,456,789.00	Multi Value	£-10	-10 kg.	No Value

Compare Combination (class="tbl-combo-compare")

Name	Testing Tester	Foo Bar
Urk	http://twigkit.com	http://twigkit.com
Date	Tuesday, November 11, 2014	Wednesday, November 11, 2015
Number	$123,456,789.00	-$123,456,789.00
multi	Multi Value	Multi Value
Prefix	£10	£-10
Suffix	10 kg.	-10 kg.
Empty	No Value	No Value

table:row

Description

Creates a table row `<tr>`

Usage

as element:

```
<table:row> </table:row>
```

task:form

Description

Posts form data to create a Twigkit Task. Any form fields with a name prefixed with `attr-` will be added as attributes to the created task.

State classes of `submitting`, `success`, and `error` are added to the form element to allow the UI to be customised.

Usage

as element:

```
<task:form task="{string}" [redirect="{string}"]> </task:form>
```

Parameters

Param	Type	Details
task	**string**	The task name. For example, if a configuration logger.conf is found under resources/conf/task/, the task name would be task.logger
redirect *(optional)*	**string**	If set, the form will redirect to this URL on successful submission

task:list

Description

Displays the current user's task list and regularly checks for updates. Any HTML within the tag will have access to a `task` variable for the current task being iterated over in the list. As well as the `stop`, `start`, and `delete` methods for interacting with the task.

Usage

as element:

```
<task:list [empty-text="{string}"] [interval="{Number}"] [date-format="{string}"]
[show-expiry="{string}"]> </task:list>
```

Parameters

Param	Type	Details
emptyText *(optional)*	**string**	The message to show if the user has no tasks. Default: "You have no tasks."
interval *(optional)*	**Number**	How often statues updates should be requested in milliseconds. Default 1000. Set to false to disable.
date-format *(optional)*	**string**	Date format to display all dates in. Default 'medium'.
show-expiry *(optional)*	**string**	Date format to display all dates in. Default 'medium'.

toggle:controls

Description

The toggle controls directive

Usage

as element:

```
<toggle:controls labels="{string}" panes="{string}" [title="{string}"]>
</toggle:controls>
```

Directive info

- This directive creates new scope.

Parameters

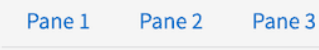
Param	Type	Details
labels	**string**	Labels that describe each corresponding pane. There must be exactly the same number of labels as there are panes.
panes	**string**	This parameter associates the controls with the corresponding panes by referencing the panes' id parameter.
title *(optional)*	**string**	The title of the pane controls.

Example

Source

```
<toggle:controls labels="Pane 1, Pane 2, Pane 3" panes="panes"></toggle:controls>
<toggle:panes id="panes"> <toggle:pane> <h3 id="example_pane-1">Pane 1</h3>
</toggle:pane> <toggle:pane> <h3 id="example_pane-2">Pane 2</h3> </toggle:pane>
<toggle:pane> <h3 id="example_pane-3">Pane 3</h3> </toggle:pane> </toggle:panes>
```

Result



toggle:pane

Description

The toggle pane directive.

Usage

as element:

```
<toggle:pane [visible="{boolean}"]> </toggle:pane>
```

Directive info

- This directive creates new scope.

Parameters

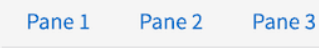
Param	Type	Details
visible *(optional)*	**boolean**	Used to denote if the pane should be shown by default.

Example

Source

```
<toggle:controls labels="Pane 1, Pane 2, Pane 3" panes="panes"></toggle:controls>
<toggle:panes id="panes"> <toggle:pane> <h3 id="example_pane-1">Pane 1</h3>
</toggle:pane> <toggle:pane> <h3 id="example_pane-2">Pane 2</h3> </toggle:pane>
<toggle:pane> <h3 id="example_pane-3">Pane 3</h3> </toggle:pane> </toggle:panes>
```

Result



toggle:panes

Description

The toggle panes directive

Usage

as element:

```
<toggle:panes id="{string}" [allow-multiple="{boolean}"]> </toggle:panes>
```

Directive info

- This directive creates new scope.

Parameters

Param	Type	Details
id	**string**	Id of the element
allow-multiple *(optional)*	**boolean**	Whether or not multiple widgets should be shown at once. Default: false.

Example

Source

```
<p>Default tabs</p> <toggle:controls labels="Pane 1, Pane 2, Pane 3" panes="panes">
</toggle:controls> <toggle:panes id="panes"> <toggle:pane> <h3 id="example_pane-
1">Pane 1</h3> </toggle:pane> <toggle:pane> <h3 id="example_pane-2">Pane 2</h3>
</toggle:pane> <toggle:pane> <h3 id="example_pane-3">Pane 3</h3> </toggle:pane>
</toggle:panes> <br> <p>Line</p> <toggle:controls labels="Pane 1, Pane 2, Pane 3"
panes="line" styling="line"></toggle:controls> <toggle:panes id="line">
<toggle:pane> <h3 id="example_pane-1">Pane 1</h3> </toggle:pane> <toggle:pane> <h3
id="example_pane-2">Pane 2</h3> </toggle:pane> <toggle:pane> <h3 id="example_pane-
3">Pane 3</h3> </toggle:pane> </toggle:panes> <br> <p>Pills</p> <toggle:controls
labels="Pane 1, Pane 2, Pane 3" panes="pills" styling="pill"></toggle:controls>
<toggle:panes id="pills"> <toggle:pane> <h3 id="example_pane-1">Pane 1</h3>
</toggle:pane> <toggle:pane> <h3 id="example_pane-2">Pane 2</h3> </toggle:pane>
<toggle:pane> <h3 id="example_pane-3">Pane 3</h3> </toggle:pane> </toggle:panes>
<br> <p>Simple</p> <toggle:controls labels="Pane 1, Pane 2, Pane 3" panes="simple"
styling="simple"></toggle:controls> <toggle:panes id="simple"> <toggle:pane> <h3
id="example_pane-1">Pane 1</h3> </toggle:pane> <toggle:pane> <h3 id="example_pane-
2">Pane 2</h3> </toggle:pane> <toggle:pane> <h3 id="example_pane-3">Pane 3</h3>
</toggle:pane> </toggle:panes> <p>Tabs Bordered</p> <toggle:controls labels="Pane
1, Pane 2, Pane 3" panes="bordered" styling="bordered"></toggle:controls>
<toggle:panes id="bordered"> <toggle:pane> <h3 id="example_pane-1">Pane 1</h3>
</toggle:pane> <toggle:pane> <h3 id="example_pane-2">Pane 2</h3> </toggle:pane>
<toggle:pane> <h3 id="example_pane-3">Pane 3</h3> </toggle:pane> </toggle:panes>
<br> <p>Line Bordered</p> <toggle:controls labels="Pane 1, Pane 2, Pane 3"
panes="line-b" styling="line bordered"></toggle:controls> <toggle:panes id="line-
b"> <toggle:pane> <h3 id="example_pane-1">Pane 1</h3> </toggle:pane> <toggle:pane>
<h3 id="example_pane-2">Pane 2</h3> </toggle:pane> <toggle:pane> <h3
id="example_pane-3">Pane 3</h3> </toggle:pane> </toggle:panes> <br> <p>Pills
Bordered</p> <toggle:controls labels="Pane 1, Pane 2, Pane 3" panes="pills-b"
styling="pill bordered"></toggle:controls> <toggle:panes id="pills-b">
<toggle:pane> <h3 id="example_pane-1">Pane 1</h3> </toggle:pane> <toggle:pane> <h3
id="example_pane-2">Pane 2</h3> </toggle:pane> <toggle:pane> <h3 id="example_pane-
3">Pane 3</h3> </toggle:pane> </toggle:panes> <br> <p>Simple Bordered</p>
<toggle:controls labels="Pane 1, Pane 2, Pane 3" panes="simple-b" styling="simple
bordered"></toggle:controls> <toggle:panes id="simple-b"> <toggle:pane> <h3
id="example_pane-1">Pane 1</h3> </toggle:pane> <toggle:pane> <h3 id="example_pane-
2">Pane 2</h3> </toggle:pane> <toggle:pane> <h3 id="example_pane-3">Pane 3</h3>
</toggle:pane> </toggle:panes>
```

Result

Default tabs

Pane 1 Pane 2 Pane 3

Line

Pane 1 Pane 2 Pane 3

Pills

Pane 1 Pane 2 Pane 3

Simple

Pane 1 Pane 2 Pane 3

Tabs Bordered

Pane 1 Pane 2 Pane 3

Line Bordered

Pane 1 Pane 2 Pane 3

Pills Bordered

Pane 1 Pane 2 Pane 3

Simple Bordered

Pane 1 Pane 2 Pane 3

tooltip

Description

Creates a tooltip.

Usage

as attribute

```
<ANY tooltip="{string}"> ... </ANY>
```

Parameters

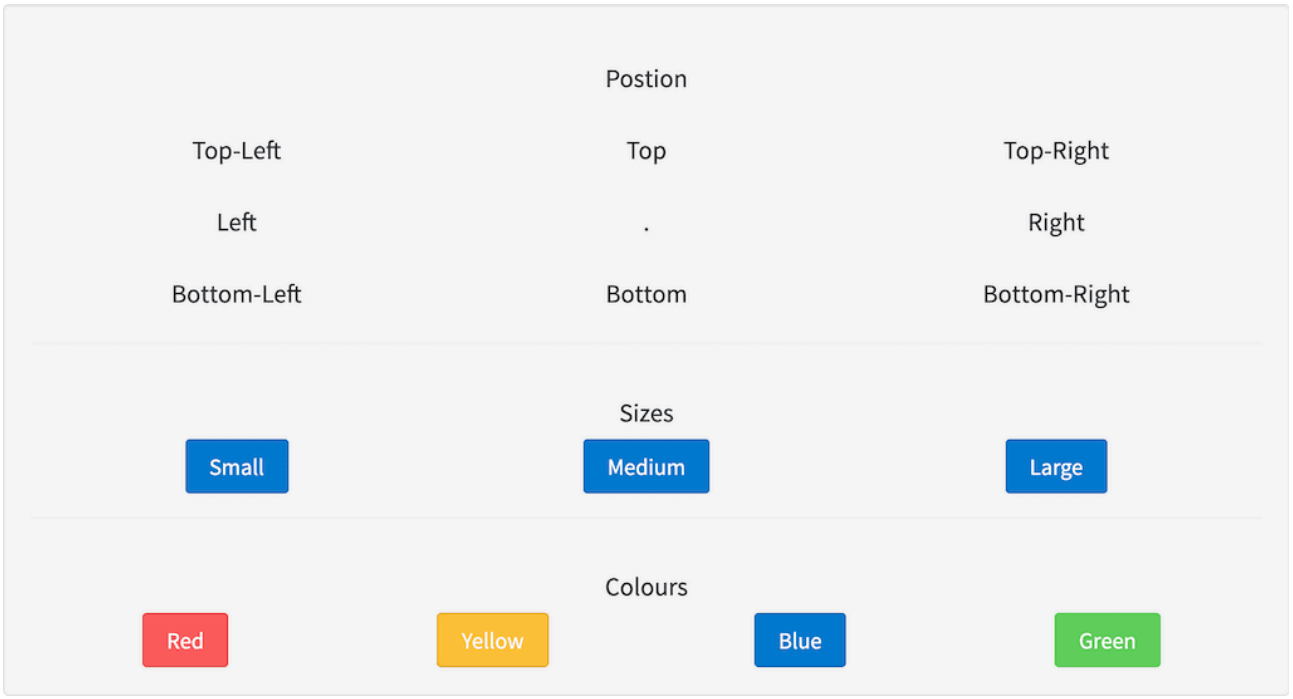
Param	Type	Details
tooltip	**string**	The text to display in the tooltip

Example

Source

```
<layout:grid styling="textcenter"> <layout:block medium="1-1"> <h3
id="example_postion">Postion</h3> </layout:block> <layout:block medium="1-3"> <h3
tooltip="Top Left" styling="tooltip-top-left" id="example_top-left">Top-Left</h3>
</layout:block> <layout:block medium="1-3"> <h3 tooltip="Top!" styling="tooltip-
top" id="example_top">Top</h3> </layout:block> <layout:block medium="1-3"> <h3
tooltip="Top Right" styling="tooltip-top-right" id="example_top-right">Top-
Right</h3> </layout:block> </layout:grid> <layout:grid styling="textcenter">
<layout:block medium="1-3"> <h3 tooltip="Left" styling="tooltip-left"
id="example_left">Left</h3> </layout:block> <layout:block medium="1-3"> <h3
id="example">.</h3> </layout:block> <layout:block medium="1-3"> <h3 tooltip="Right"
styling="tooltip-right" id="example_right">Right</h3> </layout:block>
</layout:grid> <layout:grid styling="textcenter"> <layout:block medium="1-3"> <h3
tooltip="Bottom Left" styling="tooltip-bottom-left" id="example_bottom-
left">Bottom-Left</h3> </layout:block> <layout:block medium="1-3"> <h3
tooltip="Bottom!" styling="tooltip-bottom" id="example_bottom">Bottom</h3>
</layout:block> <layout:block medium="1-3"> <h3 tooltip="Bottom Right"
styling="tooltip-bottom-right" id="example_bottom-right">Bottom-Right</h3>
</layout:block> </layout:grid> <hr /> <layout:grid styling="textcenter">
<layout:block medium="1-1"> <h3 id="example_sizes">Sizes</h3> </layout:block>
<layout:block medium="1-3"> <button tooltip="You can show very very long sentences
inside tooltips by using various available size variation classes." styling="button
tooltip-top tooltip-small">Small</button> </layout:block> <layout:block medium="1-
3"> <button tooltip="You can show very very long sentences inside tooltips by using
various available size variation classes." styling="button tooltip-top tooltip-
medium">Medium</button> </layout:block> <layout:block medium="1-3"> <button
tooltip="You can show very very long sentences inside tooltips by using various
available size variation classes." styling="button tooltip-top tooltip-
large">Large</button> </layout:block> </layout:grid> <hr /> <layout:grid
styling="textcenter"> <layout:block medium="1-1"> <h3
id="example_colours">Colours</h3> </layout:block> <layout:block medium="1-4">
<button tooltip="This is an error tooltip" styling="button bg-red tooltip-top
tooltip-error">Red</button> </layout:block> <layout:block medium="1-4"> <button
tooltip="This is a warning tooltip" styling="button bg-yellow tooltip-top tooltip-
warning">Yellow</button> </layout:block> <layout:block medium="1-4"> <button
tooltip="This is an info tooltip" styling="button tooltip-top tooltip-
info">Blue</button> </layout:block> <layout:block medium="1-4"> <button
tooltip="This is a success tooltip" styling="button button-green tooltip-top
tooltip-success">Green</button> </layout:block> </layout:grid>
```

Result


Postion
Top-Left
Top
Top-Right
Left
.
Right
Bottom-Left
Bottom
Bottom-Right
Sizes
Small
Medium
Large
Colours
Red
Yellow
Blue
Green

track:clicks

Description

Usage

as element:

```
<track:clicks [query="{query}"] [type="{string}"] [response="{string}"] [fields="
{string|expression}"] [any_attribute="{string}"]> </track:clicks>
```

Parameters

Param	Type	Details
query *(optional)*	**query**	The name of the query associated with the click.
type *(optional)*	**string**	Type of click event, (Default: link or result if click event happens in a search:result)
response *(optional)*	**string**	The name of the response to get the fact called `tracking` from so that this information can be sent to the activity endpoint. If a response is not provided and the click happens in a *Search Result List* we will get the name of the response from the result-list if applicable.
fields *(optional)*	**string]link: [expression**	If the click happens within a result send the following fields (comma separated) to the endpoint. (e.g - `name,title,url`)
any_attribute *(optional)*	**string**	You can add any custom attribute here such as `foo="bar"` and this data will be sent to the endpoint. The attribute must not start with `ng-` and also cannot be `url`,`label`,`type`,`query`,`offset`,`pos`,`fields`,`id`, `response` or `class`.

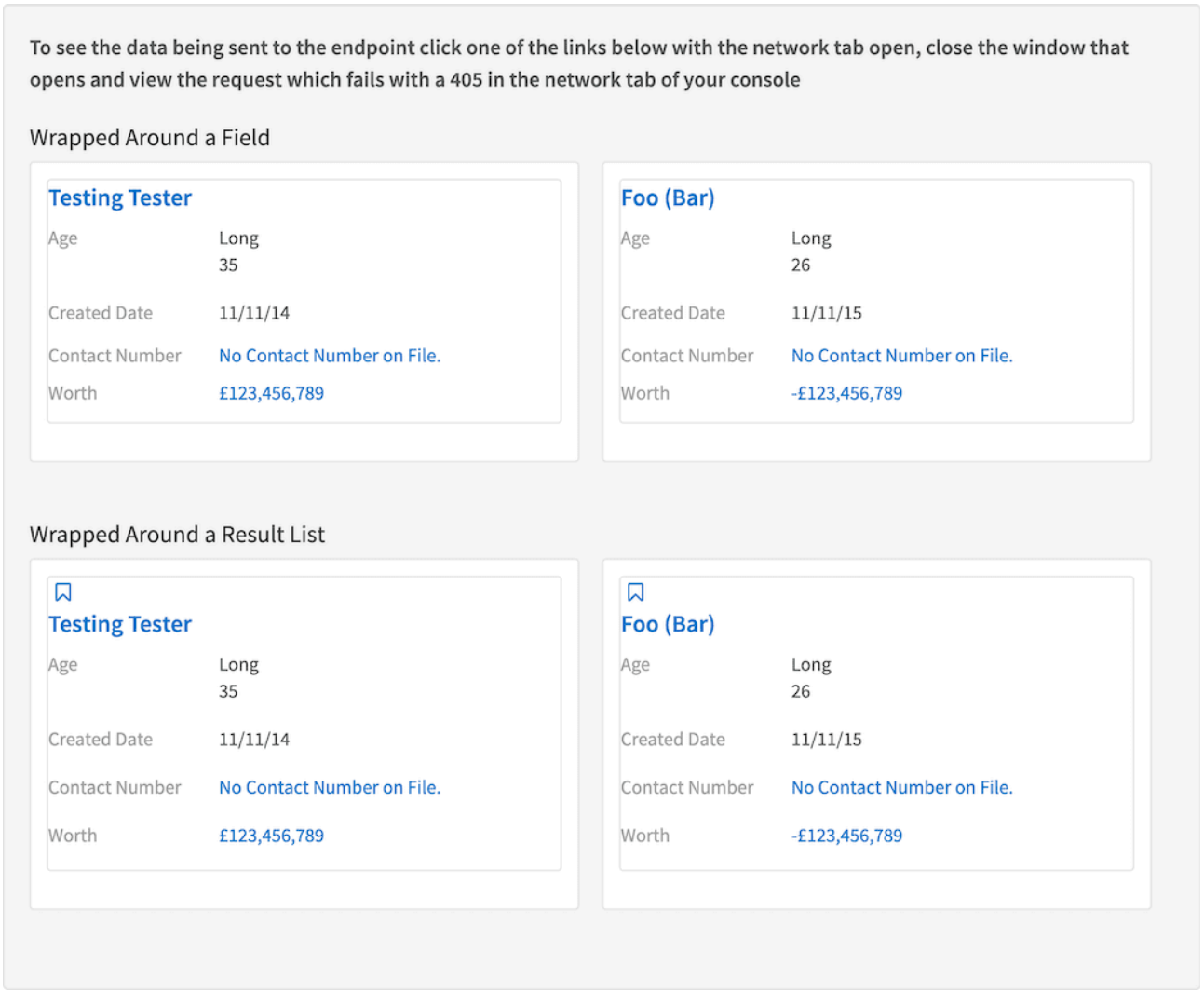
Example

Source

```
<layout:grid> <layout:block md="1-1"> <strong>To see the data being sent to the endpoint click one of the links below with the network tab open, close the window that opens and view the request which fails with a 405 in the network tab of your console </strong> <h3 id="example_wrapped-around-a-field">Wrapped Around a Field</h3> <search:result-list response="response" styling="cards-sm-1 cards-md-2"> <search:result> <layout:block> <div class="tk-stl-card"> <search:field name="name" styling="title" urlfield="url"></search:field> <search:field name="age" styling="label-left" label="Age"></search:field> <search:field name="date" styling="label-left" label="Created Date" date-format="dd/MM/yy"></search:field> <track:clicks> <search:field name="blank" styling="label-left" label="Contact Number" default-value="No Contact Number on File." url="http://twigkit.com" target="_blank"></search:field> </track:clicks> <track:clicks> <search:field name="number" styling="label-left" label="Worth" number-format="£#,###.##" url="http://twigkit.com" target="_blank"></search:field> </track:clicks> <search:field name="madeup" styling="label-left" label="Test"></search:field> </div> </layout:block> </search:result> </search:result-list> <h3 id="example_wrapped-around-a-result-list">Wrapped Around a Result List</h3> <track:clicks type="foo"> <search:result-list response="response" styling="cards-sm-1 cards-md-2"> <search:result> <layout:block> <div class="tk-stl-card"> <collaborate:bookmark></collaborate:bookmark> <search:field name="name" styling="title" urlfield="url"></search:field> <search:field name="age" styling="label-left" label="Age"></search:field> <search:field name="date" styling="label-left" label="Created Date" date-format="dd/MM/yy"></search:field> <search:field name="blank" styling="label-left" label="Contact Number" default-value="No Contact Number on File." url="http://twigkit.com" target="_blank"> </search:field> <search:field name="number" styling="label-left" label="Worth" number-format="£#,###.##" url="http://twigkit.com" target="_blank"></search:field> <search:field name="madeup" styling="label-left" label="Test"></search:field> </div> </layout:block> </search:result> </search:result-list> </track:clicks> </layout:block> </layout:grid>
```

```
angular.module('lightning') .controller('ExampleController', ['$scope','$timeout','ResponseService', function($scope,$timeout,ResponseService) { $scope.response = { page: 2, query: { rpp: 50 }, hits:{ act:['long','1234'] }, facets: { }, results: [ { result: { id:"123456abcdefgh", fields: { name: { name: 'name', val: ['Testing Tester'] }, url: { name: 'url', val: ['http://twigkit.com'] }, age: { name: 'age', val: ['Long', 35] }, date: { name: 'date', val: ['2014-11-11T00:00:00'] }, blank:{ name:'blank', val:null, }, number:{ name:'number', val: [123456789] }, }, }, },{ result: { id:"abcdefgh123456", fields: { name: { name: 'name', val: ['Foo (Bar)'] }, url: { name: 'url', val: ['http://twigkit.com'] }, age: { name: 'age', val: ['Long', 26] }, date: { name: 'date', val: ['2015-11-11T00:00:00'] }, blank:{ name:'blank', val:null, }, number:{ name:'number', val: [-123456789] } }, }, }, ], } $scope.results = $scope.response.results; ResponseService.setResponse('response', $scope.response); }]);
```

Result


To see the data being sent to the endpoint click one of the links below with the network tab open, close the window that opens and view the request which fails with a 405 in the network tab of your console
Wrapped Around a Field
Testing Tester
Age
Long
35
Created Date
11/11/14
Contact Number
No Contact Number on File.
Worth
£123,456,789
Foo (Bar)
Age
Long
26
Created Date
11/11/15
Contact Number
No Contact Number on File.
Worth
-£123,456,789
Wrapped Around a Result List
Testing Tester
Age
Long
35
Created Date
11/11/14
Contact Number
No Contact Number on File.
Worth
£123,456,789
Foo (Bar)
Age
Long
26
Created Date
11/11/15
Contact Number
No Contact Number on File.
Worth
-£123,456,789

track:query

Description

Usage

as element:

```
<track:query [var="{string}"] [host="{string}"] [page-title="{string}"] [path="{string}"] [query="{string}"]> </track:query>
```

Directive info

- This directive creates new scope.

Parameters

Param	Type	Details
var *(optional)*	**string**	The variable to assign the query to. Default:'queryToken'
host *(optional)*	**string**	Endpoint where the twigkit instance is to send the data to.
page-title *(optional)*	**string**	Title of the page where the query is sent to. (Defaults to current page title).
path *(optional)*	**string**	Relative path of the page where the query is sent to. (Defaults to current URL).
query *(optional)*	**string**	The name of the query to send to the server, defaults to parameters in the URL if no query is entered.

trigger:event

Description

Triggers $rootScope event on click

Usage

as attribute

```
<ANY trigger-event="{string}" [trigger-delay="{string}"]> ... </ANY>
```

Directive info

- This directive creates new scope.

Parameters

Param	Type	Details
trigger-event	**string**	Name of the event to trigger
trigger-delay *(optional)*	**string**	Milliseconds to delay event trigger

url:tabs

Description

The Url Tabs Directive by clicking on a tab will not add the filter to the URL it will redirect the user to '/action/tab_value' if the action attribute is present on the tag. If not it will just redirect to '/tab_value'.

Usage

as element:

```
<url:tabs response="{object}" tabs="{string}" [field-name="{string}"] [all-label="{string}"] [platform="{string}"] [query="{string}"] [count-number="{string}"] [action="{string}"]> </url:tabs>
```

Directive info

- This directive creates new scope.

Parameters

Param	Type	Details
response	**object**	A response object
tabs	**string**	The tabs to output. Specified as a comma separated list. When using filter tabs, you can either specify the actual filter name, or the actual filter name with an alternative display name. To substitute actual filter names with alternative display names use the following notation: 'programme=Programmes'.
field-name *(optional)*	**string**	The field to apply filters to when in 'filter' mode.
all-label *(optional)*	**string**	Label for the option to turn off all filters. If this attribute is specified, an additional link will be displayed to show all Results (e.g. not applying a Filter).
platform *(optional)*	**string**	The Platform to use when submitting the facet without any filters applied. This is used to to get the filter counts for the original query after a user clicks on a filter (e.g. drills down).
query *(optional)*	**string**	The name of the query object used to get the count values.
count-number *(optional)*	**string**	Specify how the number representing the count should be formatted. Optional values are 'plain', 'formatted' and 'rounded'.
action *(optional)*	**string**	Url to redirect.

Events

tab_changed

When the tab is changed on this directive an event called tab_changed will be fired with the name of the tab that is now active.

This event can be listened to like so.

```
$scope.$on('tab_changed', function(e, tab){
     console.log('New Tab = ' + tab);
});
```

Type:

broadcast

Target:

root scope

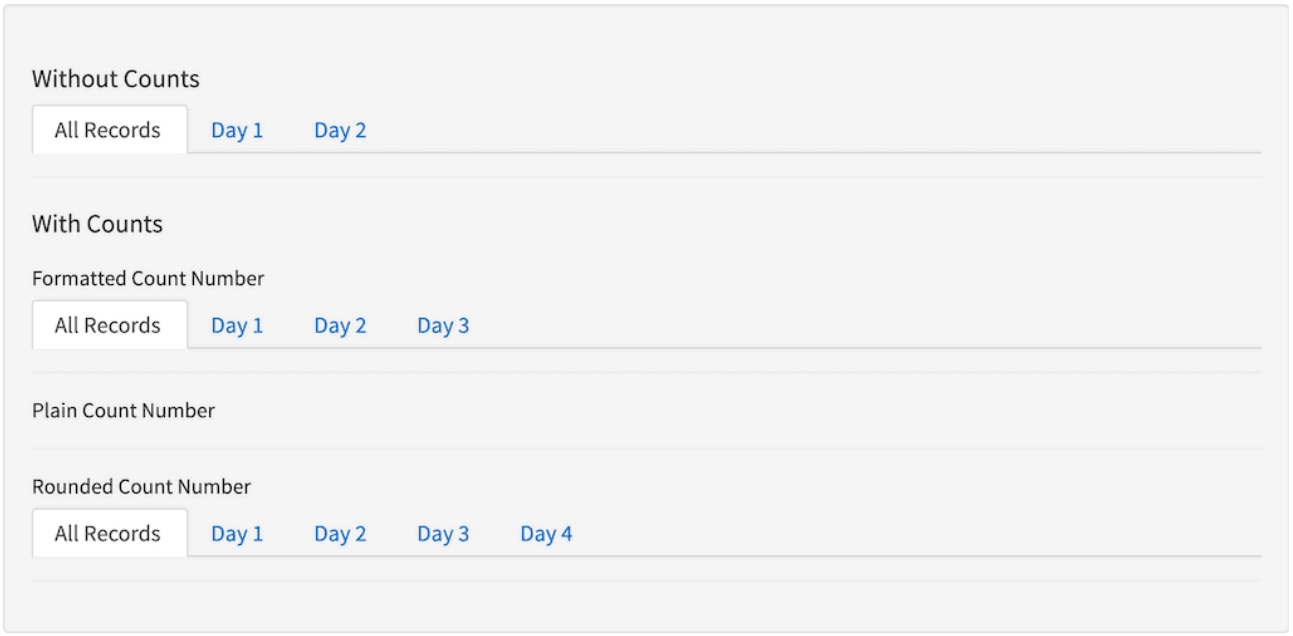
Example

Source

```
<search:platform var="platform" conf="platforms.test"></search:platform>
<search:query var="query" parameters='*'></search:query> <h3 id="example_without-
counts">Without Counts</h3> <url:tabs response="response" tabs="Day 1,Day 2" field-
name="test1" all-label="All Records"></url:tabs> <hr /> <h3 id="example_with-
counts">With Counts</h3> <h4 id="example_with-counts_formatted-count-
number">Formatted Count Number</h4> <url:tabs response="response" tabs="Day 1,Day
2,Day 3" field-name="test1" all-label="All Records" platform="platform"
query="query" count-number="formatted"></url:tabs> <hr /> <h4 id="example_with-
counts_plain-count-number">Plain Count Number</h4> <urlrch:tabs response="response"
tabs="Day 1,Day 2,Day 3" field-name="test1" all-label="All Records"
platform="platform" query="query" count-number="plain"></urlrch:tabs> <hr /> <h4
id="example_with-counts_rounded-count-number">Rounded Count Number</h4> <url:tabs
response="response" tabs="Day 1,Day 2,Day 3,Day 4" field-name="test1" all-
label="All Records" platform="platform" query="query" count-number="rounded">
</url:tabs> <hr />
```

```
angular.module('lightning') .controller('ExampleController',
['$scope','$timeout','ResponseService', function($scope,$timeout,ResponseService) {
$scope.response = { page: 2, query: { rpp: 50 }, facets: { test1: { id:'test1',
display:'test1', field:'test1', filters: [ { val: { dsp: 'Day 1', act: 'Day 1', },
count: 1000, }, { val: { dsp: 'Day 2', act: 'Day 2', }, count: 2000, }, { val: {
dsp: 'Day 3', act: 'Day 3', }, count: 2990, }, { val: { dsp: 'Day 4', act: 'Day 4',
}, count: 0, } ] }, test2: { id:'test2', display:'test2', field:'test2', filters: [
{ val: { dsp: 'Day 1', act: 'Day 1', }, count: 10, }, { val: { dsp: 'Day 2', act:
'Day 2', }, count: 20, } ] }, empty: { id:'empty', display:'Empty', field:'empty',
filters: [] }, }, results: [] } ResponseService.setResponse('response',
$scope.response); }]);
```

Result


Without Counts
All Records
Day 1
Day 2
With Counts
Formatted Count Number
All Records
Day 1
Day 2
Day 3
Plain Count Number
Rounded Count Number
All Records
Day 1
Day 2
Day 3
Day 4

widget:carousel

Description

Creates a horizontal carousel

Usage

as element:

```
<widget:carousel [carousel-name="{string}"] [carousel-watch="{Object}"] [loop="
{boolean}"] [auto-play="{number}"] [random-start="{boolean}"] [controls="
{boolean}"] [height="{number}"] [pagination="{boolean}"]> </widget:carousel>
```

as attribute

```
<ANY widget-carousel [carousel-name="{string}"] [carousel-watch="{Object}"] [loop="
{boolean}"] [auto-play="{number}"] [random-start="{boolean}"] [controls="
{boolean}"] [height="{number}"] [pagination="{boolean}"]> ... </ANY>
```

Parameters

Param	Type	Details
carousel-name *(optional)*	**string**	The name of the carousel so it can be retrieved from the Carousel service. This can be used to control the carousel from another element.
carousel-watch *(optional)*	**Object**	A variable to watch for changes to the slides data. When creating the slides in an ngRepeat the object should be set here to watch for changes.
loop *(optional)*	**boolean**	Whether the carousel should loop. Default true.
auto-play *(optional)*	**number**	Milliseconds to wait until animating to the next slide. By default the carousel will not auto slide
random-start *(optional)*	**boolean**	Randomises starting slide
controls *(optional)*	**boolean**	Adds left and right arrows for navigation. Default true.
height *(optional)*	**number**	Set the height of the carousel. By default the height is dynamically set to the height of the active slide.
pagination *(optional)*	**boolean**	Whether to show pagination dots. Default false.

Example

Source

```
<widget:carousel> <slidecontainer> <slide class="demo">Slide 1</slide> <slide class="demo">Slide 2</slide> <slide class="demo">Slide 3</slide> </slidecontainer> </widget:carousel>
```

```
.demo { min-height: 300px; text-align: center; margin-top: 140px; } .tk-stl-carousel { display: block; width: 100%; height: 300px; white-space: nowrap; position: relative; overflow: hidden; transition: height .2s ease } .tk-stl-carousel slidecontainer { display: block; position: absolute; top: 0; width: 100%; height: 100%; z-index: 1; font-size: 0 } .tk-stl-carousel slidecontainer.carousel-ignore-first-slide { -webkit-transform: translate(-100%, 0); -ms-transform: translate(-100%, 0); transform: translate(-100%, 0) } .tk-stl-carousel slidecontainer.carousel-animate { transition: -webkit-transform .5s ease-out 0s; transition: transform .5s ease-out 0s; transition: transform .5s ease-out 0s, -webkit-transform .5s ease-out 0s } .tk-stl-carousel slidecontainer slide { position: relative; display: inline-block; vertical-align: top; font-size: 16px; width: 100%; height: 100%; background-size: cover; background-position: center } .tk-stl-carousel slidecontainer slide img { pointer-events: none; margin: 0 auto; display: block } .tk-stl-carousel .carousel-arrow { display: inline-block; display: -webkit-flex; display: -ms-flexbox; display: flex; -webkit-justify-content: center; -ms-flex-pack: center; justify-content: center; -webkit-align-items: center; -ms-flex-align: center; align-items: center; position: absolute; top: 0; bottom: 0; z-index: 1; width: 100px; text-align: center; cursor: pointer } .tk-stl-carousel .carousel-arrow.carousel-arrow-left { left: 0 } .tk-stl-carousel .carousel-arrow.carousel-arrow-right { right: 0 } .tk-stl-carousel .carousel-arrow .icon-angle-left, .tk-stl-carousel .carousel-arrow .icon-angle-right { font-size: 2.5em }
```

Result

widget:date-range

Description

Creates two date-time picker inputs for setting a date range filter.

Usage

as element:

```
<widget:date-range field="{string}" [date-format="{string}"] [from-label="{string}"] [to-label="{string}"] [on-set="{function}"] [min-view="{string}"] [from-model="{string}"] [to-model="{string}"]> </widget:date-range>
```

Parameters

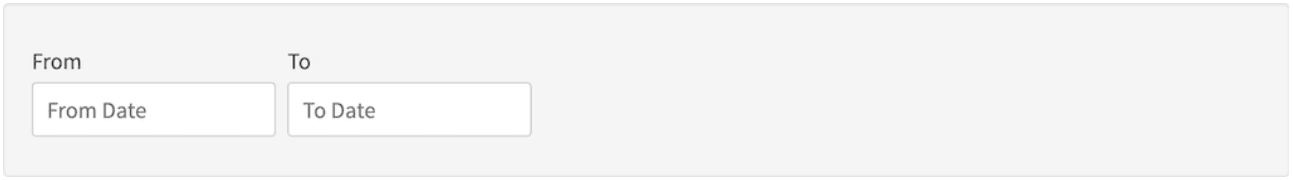
Param	Type	Details
field	**string**	Field to apply filter to
date-format *(optional)*	**string**	Date format to display
from-label *(optional)*	**string**	From input label
to-label *(optional)*	**string**	To input label
on-set *(optional)*	**function**	Function to call when a date has been set instead of applying filters. Two arguments are passed to the function from Date object, to Date object
min-view *(optional)*	**string**	The minimum view of the date picker to display. Values: 'year/month/day/hour/minute'. Default 'minute'.
from-model *(optional)*	**string**	A binding to a scoped variable, the from date picker will be set using this value.
to-model *(optional)*	**string**	A binding to a scoped variable, the to date picker will be set using this value.

Example

Source

```
<widget:date-range field="date"></widget:date-range>
```

Result



widget:datetimepicker

Description

The Header Search directive

Usage

as element:

```
<widget:datetimepicker ng-model="{Object}" [on-set-time="{Object}"] [datepicker-config="{Object}"] date-from="{String}"> </widget:datetimepicker>
```

Parameters

Param	Type	Details
ngModel	**Object**	Scope variable to update
onSetTime *(optional)*	**Object**	Function to call when the date/time has been selected
datepickerConfig *(optional)*	**Object**	Config for datepicker. \{minView: 'year/month/day/hour'}
dateFrom	**String**	Earliest pickable date, updates automatically when changed.

Example

Source

```
<input type="text" ng-model="data | date:'fullDate'" /> <widget:datetimepicker data-ng-model="data" data-on-set-time="onTimeSet(newDate, oldDate)"> </widget:datetimepicker>
```

```
angular.module('lightning') .controller('ExampleController', ['$scope', function($scope) { $scope.data = new Date(); $scope.onTimeSet = function (newDate, oldDate) { console.log(newDate, oldDate); } }]);
```

Result

Friday, May 21, 2021

widget:file-hierarchy

Description

Creates a horizontal column view for a file hierarchy

Usage

as element:

```
<widget:file-hierarchy files="{mixed}" on-submit="{function}" button-label="
{string}"> </widget:file-hierarchy>
```

Parameters

Param	Type	Details
files	**mixed**	The hierarchy of files
on-submit	**function**	A function to call when the submit button is clicked
button-label	**string**	Label for the button, button only shows if a label is not empty

widget:icon

Description

Creates an icon

Usage

as element:

```
<widget:icon icon="{string}"> </widget:icon>
```

Parameters

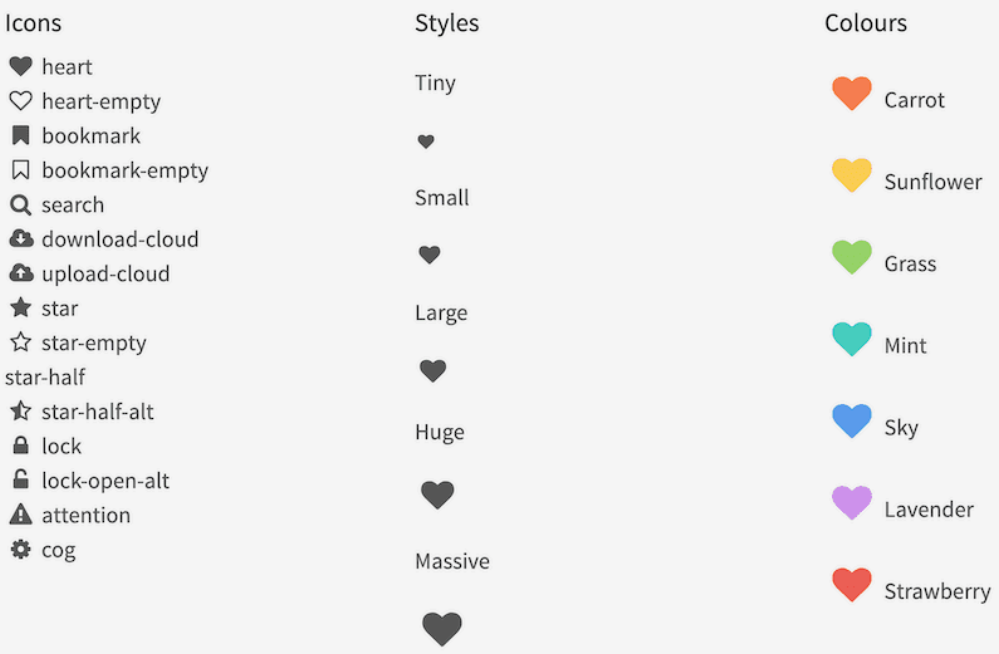
Param	Type	Details
icon	**string**	The name of the icon to display.

Example

Source

```
<layout:grid> <layout:block medium="1-3"> <h3 id="example_icons">Icons</h3>
<widget:icon icon="heart"></widget:icon> heart<br /> <widget:icon icon="heart-
empty"></widget:icon> heart-empty<br /> <widget:icon icon="bookmark"></widget:icon>
bookmark<br /> <widget:icon icon="bookmark-empty"></widget:icon> bookmark-empty<br
/> <widget:icon icon="search"></widget:icon> search<br /> <widget:icon
icon="download-cloud"></widget:icon> download-cloud<br /> <widget:icon
icon="upload-cloud"></widget:icon> upload-cloud<br /> <widget:icon icon="star">
</widget:icon> star<br /> <widget:icon icon="star-empty"></widget:icon> star-
empty<br /> <widget:icon icon="star-half"></widget:icon> star-half<br />
<widget:icon icon="star-half-alt"></widget:icon> star-half-alt<br /> <widget:icon
icon="lock"></widget:icon> lock<br /> <widget:icon icon="lock-open-alt">
</widget:icon> lock-open-alt<br /> <widget:icon icon="attention"></widget:icon>
attention<br /> <widget:icon icon="cog"></widget:icon> cog<br /> </layout:block>
<layout:block medium="1-3"> <h3 id="example_styles">Styles</h3> <p>Tiny</p>
<widget:icon icon="heart" style="tiny"></widget:icon> <br /> <p>Small</p>
<widget:icon icon="heart" style="small"></widget:icon> <br /> <p>Large</p>
<widget:icon icon="heart" style="large"></widget:icon> <br /> <p>Huge</p>
<widget:icon icon="heart" style="huge"></widget:icon> <br /> <p>Massive</p>
<widget:icon icon="heart" style="massive"></widget:icon> <br /> </layout:block>
<layout:block medium="1-3"> <h3 id="example_colours">Colours</h3> <p> <widget:icon
icon="heart" style="massive text-carrot"></widget:icon> Carrot </p> <p>
<widget:icon icon="heart" style="massive text-sunflower"></widget:icon> Sunflower
</p> <p> <widget:icon icon="heart" style="massive text-grass"></widget:icon> Grass
</p> <p> <widget:icon icon="heart" style="massive text-mint"></widget:icon> Mint
</p> <p> <widget:icon icon="heart" style="massive text-sky"></widget:icon> Sky </p>
<p> <widget:icon icon="heart" style="massive text-lavender"></widget:icon> Lavender
</p> <p> <widget:icon icon="heart" style="massive text-strawberry"></widget:icon>
Strawberry </p> </layout:block> </layout:grid>
```

Result



widget:in-view

Description

Wrap around any element and it will only be loaded when it is in view.

Usage

as element:

```
<widget:in-view> </widget:in-view>
```

widget:license

Description

Shows the Twigkit license expiration date. When transcluded the following scoped variables are available: (bool) licenseExpired (int) licenseDaysRemaining (string) licenseExpiryDate (bool) show

Usage

as element:

```
<widget:license [days-left="{string}"] [date-format="{string}"]> </widget:license>
```

Directive info

- This directive creates new scope.

Parameters

Param	Type	Details
daysLeft *(optional)*	**string**	The number of days left on the license. Also excepts numbers in the format <5 or >5 to specify a range
dateFormat *(optional)*	**string**	The date format to show the expiration date in

widget:login-form

Description

Creates a login form

Usage

as element:

```
<widget:login-form [logo="{string}"] [logo-alt="{string}"] [title="{string}"]
[message="{string}"] [action="{string}"] [method="{string}"] [username="{string}"]
[password="{string}"] [username-placeholder="{string}"] [password-placeholder="
{string}"] [remember="{string}"] [submit-label="{string}"] [access-denied="
{string}"] [error-message="{string}"] [reset-link="{string}"] [create-link="
{string}"] [username-error-message="{string}"] [password-error-message="{string}"]>
</widget:login-form>
```

Parameters

Param	Type	Details
logo *(optional)*	**string**	The path to the logo
logo-alt *(optional)*	**string**	The alt text for the logo
title *(optional)*	**string**	The title text that displays below the logo
message *(optional)*	**string**	The message to display below the title
action *(optional)*	**string**	URL to send the form data to
method *(optional)*	**string**	The form method to use. Default POST
username *(optional)*	**string**	The name of the the username input
password *(optional)*	**string**	The name of the the password input
username-placeholder *(optional)*	**string**	The placeholder of the the username input
password-placeholder *(optional)*	**string**	The placeholder of the the password input
remember *(optional)*	**string**	The name of the the remember input. Set to false to hide remember me checkbox *(default: "remember")*
submit-label *(optional)*	**string**	The text on the submit button. *(default: "Login")*
access-denied *(optional)*	**string**	Checks the URL for the value of access-denied, if found the error message will be displayed E.g. access=denied
error-message *(optional)*	**string**	The error message to display when the login is rejected. *(default: "Invalid username or password")*

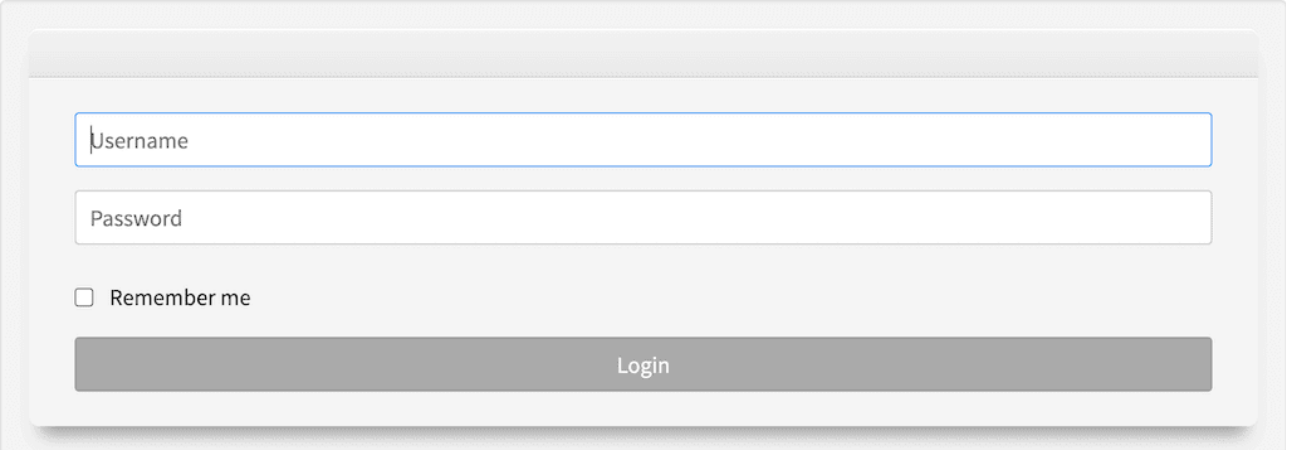
Param	Type	Details
reset-link *(optional)*	**string**	Provide a link to reset the user's password
create-link *(optional)*	**string**	Provide a link to for a user to request an account
username-error-message *(optional)*	**string**	Message to display if the username is not entered or valid. *(default: "This field is required.")*
password-error-message *(optional)*	**string**	Message to display if the password is not entered or valid. *(default: "This field is required.")*

Example

Source

```
<widget:login-form></widget:login-form>
```

Result



widget:login-page

Description

Creates a login page with a form. All of the attributes are passed to the form.

Usage

as element:

```
<widget:login-page [logo="{string}"] [logo-alt="{string}"] [title="{string}"]
[message="{string}"] [action="{string}"] [method="{string}"] [username="{string}"]
[password="{string}"] [username-placeholder="{string}"] [password-placeholder="
{string}"] [remember="{string}"] [submit-label="{string}"] [access-denied="
{string}"] [error-message="{string}"]> </widget:login-page>
```

Parameters

Param	Type	Details
logo *(optional)*	**string**	The path to the logo
logoAlt *(optional)*	**string**	The alt text for the logo
title *(optional)*	**string**	The title text that displays below the logo
message *(optional)*	**string**	The message to display below the title
action *(optional)*	**string**	URL to send the form data to
method *(optional)*	**string**	The form method to use. Default POST
username *(optional)*	**string**	The name of the username input
password *(optional)*	**string**	The name of the password input
username-placeholder *(optional)*	**string**	The placeholder of the username input
password-placeholder *(optional)*	**string**	The placeholder of the password input
remember *(optional)*	**string**	The name of the remember input. Default: remember. Set to false to hide remember me checkbox *(default: "remember")*
submit-label *(optional)*	**string**	The text on the submit button. Default: Login *(default: "Login")*
access-denied *(optional)*	**string**	Checks the URL for the value of access-denied, if found the error message will be displayed E.g. access=denied

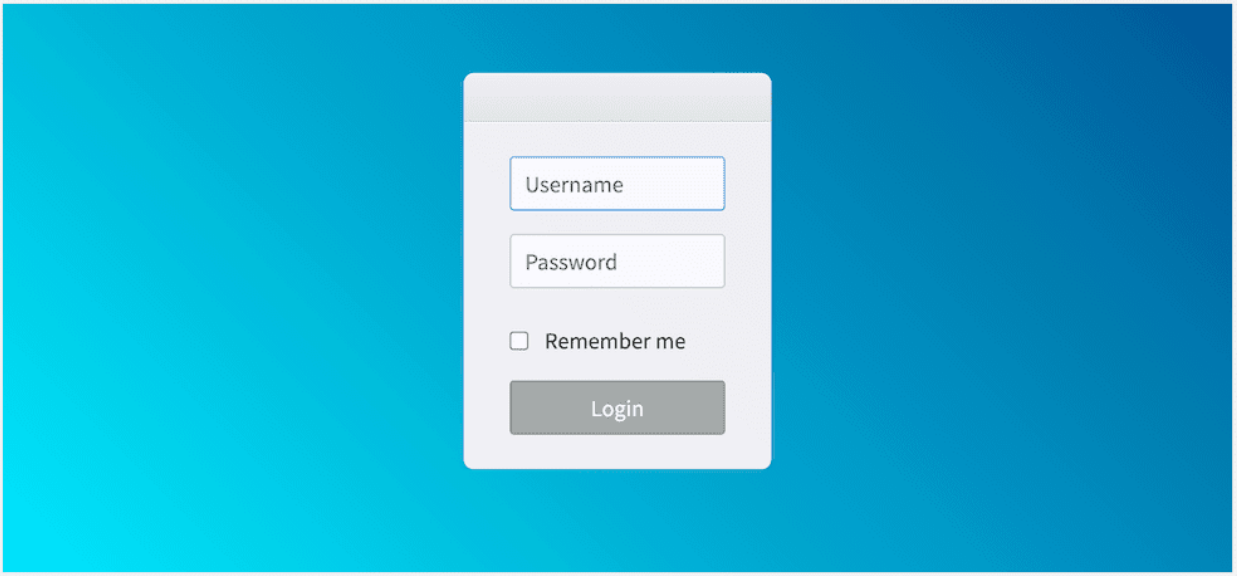
Param	Type	Details
error-message *(optional)*	**string**	The error message to display when the login is rejected. Default: Invalid username or password *(default: "Invalid username or password")*

Example

Source

```
<widget:login-page></widget:login-page>
```

Result



widget:logout

Description

Redirects the user to `<app context>/logout/` when clicked

Usage

as element:

```
<widget:logout> </widget:logout>
```

Directive info

- This directive creates new scope.

Example

Source

```
<widget:logout></widget:logout>
```

Result

Logout

widget:modal

Description

Creates a modal. The directive is automatically moved to the body so it works consistently.

Usage

as element:

```
<widget:modal id="{string}" [title="{string}"] [modal-classes="{string}"]>
</widget:modal>
```

Parameters

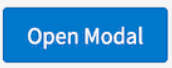
Param	Type	Details
id	**string**	An ID that can be used to trigger the modal to open
title *(optional)*	**string**	Title for the modal
modal-classes *(optional)*	**string**	Add classes to the modal elements

Example

Source

```
<button class="tk-stl-button" widget-open-modal="id">Open Modal</button>
<widget:modal title="Hello World" id="id"> <search:platform var="platform"
conf="platorms.solr"></search:platform> <search:query var="query" query="docs">
</search:query> <search:response var="response" query="query" platform="platform">
</search:response> <p>Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed
do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim
veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo
consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum
dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident,
sunt in culpa qui officia deserunt mollit anim id est laborum.</p> </widget:modal>
```

Result



widget:open-modal

Description

Applies a class to the target when toggled on.

Usage

as attribute

```
<ANY widget-open-modal="{string}"> ... </ANY>
```

Directive info

- This directive creates new scope.

Parameters

Param	Type	Details
widget-open-modal	**string**	The ID of the modal to open

Example

Source

```
<button class="tk-stl-button" widget-open-modal="id">Open Modal</button>
<widget:modal title="Hello World" id="id"> <p>Lorem ipsum dolor sit amet,
consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore
magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris
nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in
voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint
occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id
est laborum.</p> </widget:modal>
```

Result

widget:popover

Description

The popover widget allows the popover content to be displayed in a popup when the popover trigger is clicked.

Usage

as element:

```
<widget:popover [direction="{string}"] [mode="{string}"] [close-on-click="{string}"] [close-on-mouseleave="{string}"] [on-close="{function}"] [hover-timeout="{number}"]> </widget:popover>
```

Directive info

- This directive creates new scope.

Parameters

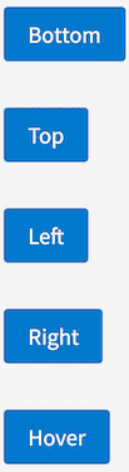
Param	Type	Details
direction *(optional)*	**string**	Where the popover should appear (top, left, right, bottom, bottom-left, bottom-center-left, bottom-center-right). Default: bottom
mode *(optional)*	**string**	How the popover should open (hover or click). Default: click
close-on-click *(optional)*	**string**	Close the popover when the popover is clicked on. Default: false
close-on-mouseleave *(optional)*	**string**	Close the popover when the mouse leaves the element.
on-close *(optional)*	**function**	Function to call when the popover is closed.
hover-timeout *(optional)*	**number**	A number to determining the number of milliseconds the popover should be displayed for after the user leaves the popover-trigger. (Only applicable to hover mode.) Default: 1000

Example

Source

```
<layout:box> <widget:popover direction="bottom" styling="popover-medium"> <popover-
trigger> <button styling="button">Bottom</button> </popover-trigger> <popover-
content> <h3 id="example_title">Title</h3> <p>Lorem ipsum dolor sit amet,
consectetur adipisicing elit. Consequuntur cum distinctio dolores enim esse fuga
incidunt laboriosam maxime nam odio pariatur provident, quae quam quidem quod saepe
voluptas. Maiores, obcaecati.</p> </popover-content> </widget:popover>
</layout:box> <layout:box> <widget:popover direction="top" styling="popover-small">
<popover-trigger><button styling="button">Top</button></popover-trigger> <popover-
content> <h3 id="example_title">Title</h3> <p>Lorem ipsum dolor sit amet,
consectetur adipisicing elit. Consequuntur cum distinctio dolores enim esse fuga
incidunt laboriosam maxime nam odio pariatur provident, quae quam quidem quod saepe
voluptas. Maiores, obcaecati.</p> </popover-content> </widget:popover>
</layout:box> </div> <layout:box> <widget:popover direction="left"> <popover-
trigger><button styling="button">Left</button></popover-trigger> <popover-content>
<h3 id="example_title">Title</h3> <p>Lorem ipsum dolor sit amet, consectetur
adipisicing elit. Consequuntur cum distinctio dolores enim esse fuga incidunt
laboriosam maxime nam odio pariatur provident, quae quam quidem quod saepe
voluptas. Maiores, obcaecati.</p> </popover-content> </widget:popover>
</layout:box> <layout:box> <widget:popover direction="right" styling="popover-
large"> <popover-trigger><button styling="button">Right</button></popover-trigger>
<popover-content> <h3 id="example_title">Title</h3> <p>Lorem ipsum dolor sit amet,
consectetur adipisicing elit. Consequuntur cum distinctio dolores enim esse fuga
incidunt laboriosam maxime nam odio pariatur provident, quae quam quidem quod saepe
voluptas. Maiores, obcaecati.</p> </popover-content> </widget:popover>
</layout:box> <layout:box> <widget:popover mode="hover"> <popover-trigger><button
styling="button">Hover</button></popover-trigger> <popover-content> <h3
id="example_hover">Hover</h3> <p>Lorem ipsum dolor sit amet, consectetur
adipisicing elit. Accusantium aliquam debitis deserunt eum minus molestiae mollitia
perspiciatis possimus provident quae. Architecto corporis dicta ea harum odio saepe
suscipit, totam ut!</p> </popover-content> </widget:popover> </layout:box>
```

Result



widget:scroll-to

Description

When clicked scrolls to the top of the page or to the specified element by ID. Styles can be added to fix the scroll to top to the bottom left right or center of the page by adding the style tag with 'bottom-right' OR 'bottom-center' OR 'bottom-left'

Usage

as element:

```
<widget:scroll-to [scroll-to-id="{string}"]> </widget:scroll-to>
```

Directive info

- This directive creates new scope.

Parameters

Param	Type	Details
scroll-to-id *(optional)*	**string**	The id of the element to scroll to.

widget:slide-down

Description

The toggle panes directive

Usage

as attribute

```
<ANY widget-slide-down [duration="{string}"] [easing="{string}"] [closed="
{boolean}"] [height="{Number}"] [response="{Object}"]> ... </ANY>
```

as class

```
<ANY class="widget-slide-down [duration: {string};] [easing: {string};] [closed:
{boolean};] [height: {Number};] [response: {Object};]"> ... </ANY>
```

Directive info

- This directive creates new scope.

Parameters

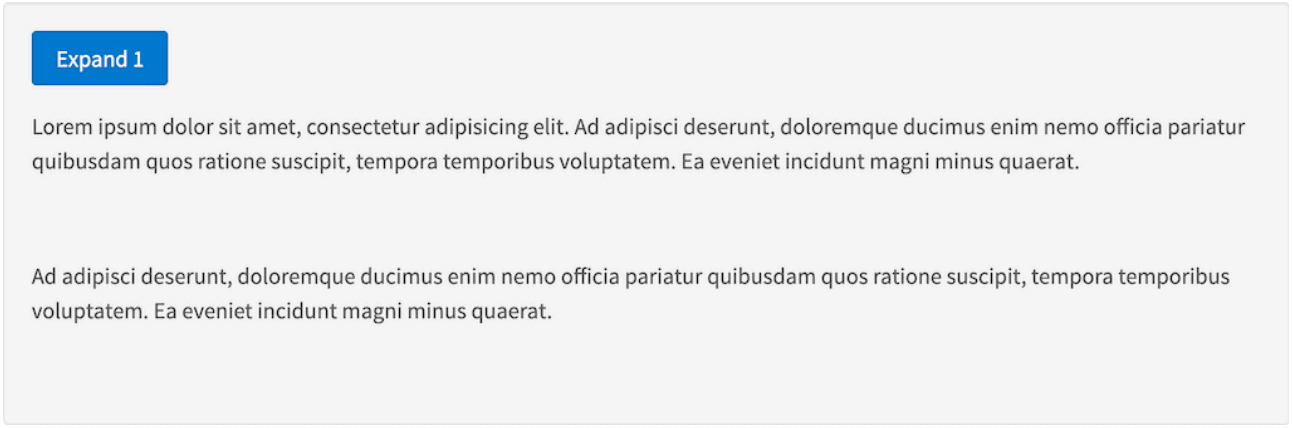
Param	Type	Details
duration *(optional)*	**string**	Duration of the animation. Default 0.3 (seconds)
easing *(optional)*	**string**	CSS timing function i.e. ease-out
closed *(optional)*	**boolean**	Starts the slidable element off as open
height *(optional)*	**Number**	Set the height of the opened panel. Useful if the content is dynamically loaded.
response *(optional)*	**Object**	Wait for a response before calculating the height and check for updates

Example

Source

```
<button class="tk-stl-button" widget-slide-toggle="#foo1">Expand 1</button> <div id="foo1" class="widget-slide-down"> <p>Lorem ipsum dolor sit amet, consectetur adipisicing elit. Ad adipisci deserunt, doloremque ducimus enim nemo officia pariatur quibusdam quos ratione suscipit, tempora temporibus voluptatem. Ea eveniet incidunt magni minus quaerat.</p> <br> <p padding-bottom="2">Ad adipisci deserunt, doloremque ducimus enim nemo officia pariatur quibusdam quos ratione suscipit, tempora temporibus voluptatem. Ea eveniet incidunt magni minus quaerat.</p> </div>
```

Result



widget:slider

Description

Creates a slider

Usage

as element:

```
<widget:slider [response="{object}"] [query="{string}"] [facet-name="{string}"]
[facet="{string}"] [field-name="{string}"] [min="{number}"] [max="{number}"]
[step="{number}"] [button-label="{string}"] [to-label="{string}"] [prefix="
{string}"] [suffix="{string}"] [invert-range="{boolean}"] [multiplier="{number}"]
[date-format="{string}"] [number-format="{string}"] [sort="{boolean}"]>
</widget:slider>
```

Parameters

Param	Type	Details
response *(optional)*	**object**	A response object
query *(optional)*	**string**	The name of the query to build links from.
facet-name *(optional)*	**string**	Facet name to use from the response object
facet *(optional)*	**string**	A facet object to display
field-name *(optional)*	**string**	The field name tells the slider which field to use when creating range filters.
min *(optional)*	**number**	The minimum value for the slider. (Defaults to 0, not required for facets)
max *(optional)*	**number**	The maximum value for the slider. (Defaults to 100, not required for facets)
step *(optional)*	**number**	The interval between each step of the slider. (Defaults to 1, not required for facets)
button-label *(optional)*	**string**	The text to be used for the 'Apply' button.
to-label *(optional)*	**string**	The string to be inserted in the filter’s display value between the 'from' and 'to' value. (Defaults to '-')
prefix *(optional)*	**string**	Add a string before the display value. Useful for currency symbols, for instance.
suffix *(optional)*	**string**	Add a string after the display value. Useful for adding units, such as “k” for thousands.
invert-range *(optional)*	**boolean**	When invertRange is set to true, the from and to values of the resulting range filter are swapped.
multiplier *(optional)*	**number**	The multiplier parameter accepts a number and multiplies the slider’s value by the multiplier when creating a filter.

Param	Type	Details
date-format *(optional)*	**string**	How to format the date. See #/api/lightning.filter:dateFormat[date format filter] for valid formats.
number-format *(optional)*	**string**	Format string for the numbers used using the Java DecimalFormat pattern. Default: ',0.'
sort *(optional)*	**boolean**	Whether the data should be sort by the directive. Defaults to true.

Example

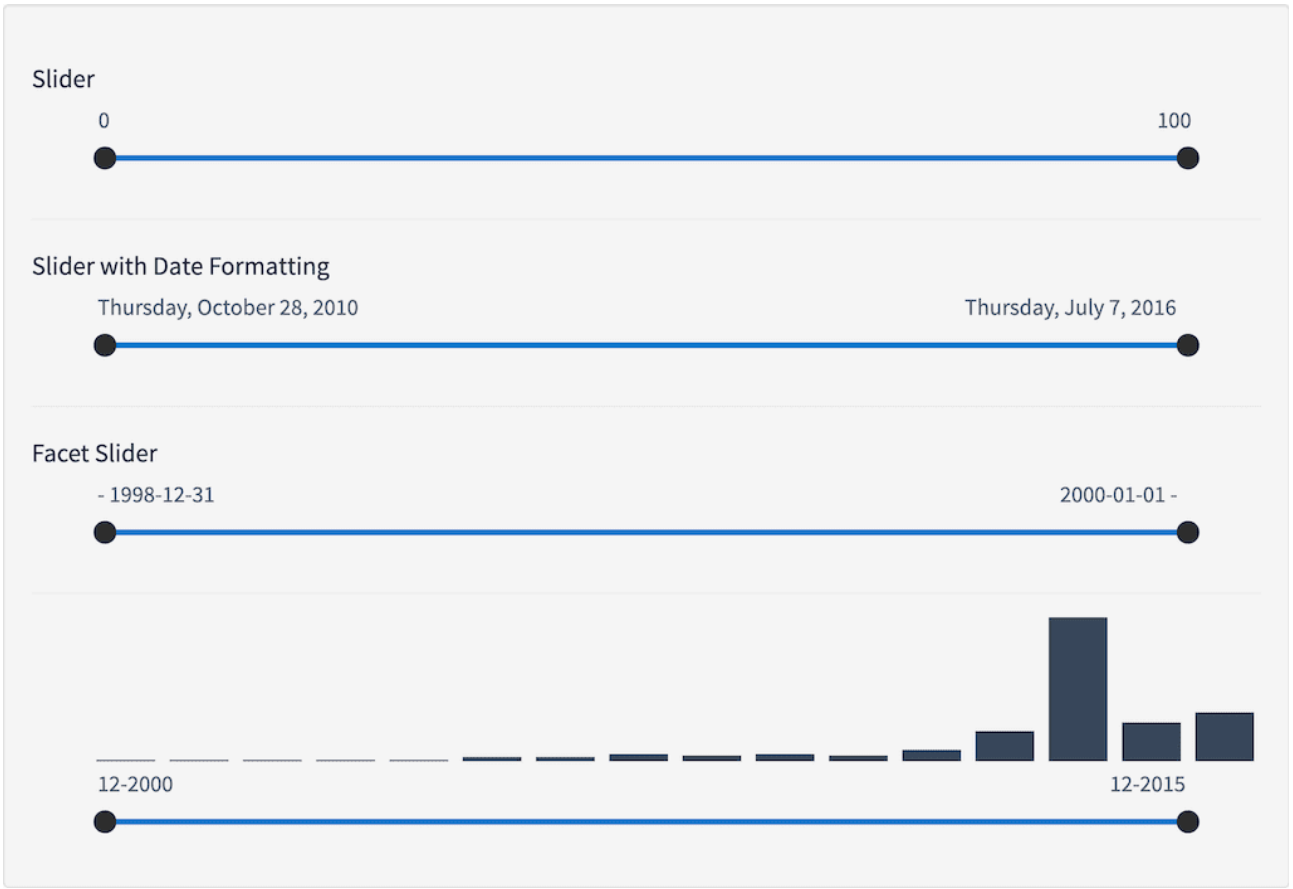
Source

```
<h3 id="example_slider">Slider</h3> <widget:slider field-name="foo" min="0"
max="100" step="1"></widget:slider> <hr /> <h3 id="example_slider-with-date-
formatting">Slider with Date Formatting</h3> <widget:slider response="response"
facet-name="test1" date-format="fullDate"></widget:slider> <hr /> <h3
id="example_facet-slider">Facet Slider</h3> <widget:slider response="response"
facet-name="test2"></widget:slider> <hr /> <chart:histogram facet="response |
facet:'Last Modified'" show="100" show-count="true" date-format="MM-yyyy">
</chart:histogram> <widget:slider response="response" facet-name="Last Modified"
date-format="MM-yyyy"></widget:slider>
```

```
angular.module('lightning') .controller('ExampleController', ['$scope','$timeout',
'ResponseService', function($scope,$timeout, ResponseService) { $scope.response = {
page: 2, query: { rpp: 50, filters:[] }, facets: { test1: { filters: [ { val: {
dsp: 1288323623006, act: 1288323623006, }, count: 10, }, { val: { dsp:
1467882419615, act: 1467882419615, }, count: 20, } ], 'max-count':20 }, test2: {
filters: [ { val: { dsp: '- 1998-12-31', act: { type:'range', to:'1998-12-
31T00:00:00Z' }, }, count: 10, }, { val: { dsp: '1999-01-01 - 1999-12-31', act: {
type:'range', from:'1999-01-01T00:00:00', to:'1999-12-31T00:00:00Z' }, }, count:
20, }, { val: { dsp: '2000-01-01 - ', act: { type:'range', from:'2000-01-
01T00:00:00Z', }, }, count: 20, } ], 'max-count':20 }, 'Last Modified':{"id":"Last
Modified","field":"Last Modified","filters":[{"field":"Last
Modified","optional":"none","match":"exact","exclude":false,"count":294,"weight":0,
"facts":{},"grp":-1,"val":{"act":{"type":"range","from":"2001-01-
01T00:00:00","to":"2002-01-
01T00:00:00","fromExcluded":false,"toExcluded":true},"dsp":"2001-01-
01T00:00:00Z"},"has-more-rel":false,"rel":[]},{"field":"Last
Modified","optional":"none","match":"exact","exclude":false,"count":403,"weight":0,
"facts":{},"grp":-1,"val":{"act":{"type":"range","from":"2002-01-
01T00:00:00","to":"2003-01-
01T00:00:00","fromExcluded":false,"toExcluded":true},"dsp":"2002-01-
01T00:00:00Z"},"has-more-rel":false,"rel":[]},{"field":"Last
Modified","optional":"none","match":"exact","exclude":false,"count":592,"weight":0,
"facts":{},"grp":-1,"val":{"act":{"type":"range","from":"2003-01-
01T00:00:00","to":"2004-01-
01T00:00:00","fromExcluded":false,"toExcluded":true},"dsp":"2003-01-
01T00:00:00Z"},"has-more-rel":false,"rel":[]},{"field":"Last
Modified","optional":"none","match":"exact","exclude":false,"count":488,"weight":0,
"facts":{},"grp":-1,"val":{"act":{"type":"range","from":"2004-01-
01T00:00:00","to":"2005-01-
01T00:00:00","fromExcluded":false,"toExcluded":true},"dsp":"2004-01-
01T00:00:00Z"},"has-more-rel":false,"rel":[]},{"field":"Last
Modified","optional":"none","match":"exact","exclude":false,"count":1610,"weight":0
,"facts":{},"grp":-1,"val":{"act":{"type":"range","from":"2005-01-
01T00:00:00","to":"2006-01-
01T00:00:00","fromExcluded":false,"toExcluded":true},"dsp":"2005-01-
01T00:00:00Z"},"has-more-rel":false,"rel":[]},{"field":"Last
Modified","optional":"none","match":"exact","exclude":false,"count":4947,"weight":0
,"facts":{},"grp":-1,"val":{"act":{"type":"range","from":"2006-01-
01T00:00:00","to":"2007-01-
01T00:00:00","fromExcluded":false,"toExcluded":true},"dsp":"2006-01-
01T00:00:00Z"},"has-more-rel":false,"rel":[]},{"field":"Last
Modified","optional":"none","match":"exact","exclude":false,"count":4058,"weight":0
,"facts":{},"grp":-1,"val":{"act":{"type":"range","from":"2007-01-
01T00:00:00","to":"2008-01-
01T00:00:00","fromExcluded":false,"toExcluded":true},"dsp":"2007-01-
01T00:00:00Z"},"has-more-rel":false,"rel":[]},{"field":"Last
Modified","optional":"none","match":"exact","exclude":false,"count":7837,"weight":0
,"facts":{},"grp":-1,"val":{"act":{"type":"range","from":"2008-01-
01T00:00:00","to":"2009-01-
01T00:00:00","fromExcluded":false,"toExcluded":true},"dsp":"2008-01-
01T00:00:00Z"},"has-more-rel":false,"rel":[]},{"field":"Last
Modified","optional":"none","match":"exact","exclude":false,"count":6011,"weight":0
,"facts":{},"grp":-1,"val":{"act":{"type":"range","from":"2009-01-
01T00:00:00","to":"2010-01-
01T00:00:00","fromExcluded":false,"toExcluded":true},"dsp":"2009-01-
01T00:00:00Z"},"has-more-rel":false,"rel":[]},{"field":"Last
Modified","optional":"none","match":"exact","exclude":false,"count":7584,"weight":0
,"facts":{},"grp":-1,"val":{"act":{"type":"range","from":"2010-01-
01T00:00:00","to":"2011-01-
01T00:00:00","fromExcluded":false,"toExcluded":true},"dsp":"2010-01-
01T00:00:00Z"},"has-more-rel":false,"rel":[]},{"field":"Last
Modified","optional":"none","match":"exact","exclude":false,"count":6284,"weight":0
,"facts":{},"grp":-1,"val":{"act":{"type":"range","from":"2011-01-
01T00:00:00","to":"2012-01-
01T00:00:00","fromExcluded":false,"toExcluded":true},"dsp":"2011-01-
```

```
01T00:00:00Z"},"has-more-rel":false,"rel":[]},{"field":"Last
Modified","optional":"none","match":"exact","exclude":false,"count":13003,"weight":
0,"facts":{},"grp":-1,"val":{"act":{"type":"range","from":"2012-01-
01T00:00:00","to":"2013-01-
01T00:00:00","fromExcluded":false,"toExcluded":true},"dsp":"2012-01-
01T00:00:00Z"},"has-more-rel":false,"rel":[]},{"field":"Last
Modified","optional":"none","match":"exact","exclude":false,"count":32742,"weight":
0,"facts":{},"grp":-1,"val":{"act":{"type":"range","from":"2013-01-
01T00:00:00","to":"2014-01-
01T00:00:00","fromExcluded":false,"toExcluded":true},"dsp":"2013-01-
01T00:00:00Z"},"has-more-rel":false,"rel":[]},{"field":"Last
Modified","optional":"none","match":"exact","exclude":false,"count":156357,"weight"
:0,"facts":{},"grp":-1,"val":{"act":{"type":"range","from":"2014-01-
01T00:00:00","to":"2015-01-
01T00:00:00","fromExcluded":false,"toExcluded":true},"dsp":"2014-01-
01T00:00:00Z"},"has-more-rel":false,"rel":[]},{"field":"Last
Modified","optional":"none","match":"exact","exclude":false,"count":42345,"weight":
0,"facts":{},"grp":-1,"val":{"act":{"type":"range","from":"2015-01-
01T00:00:00","to":"2016-01-
01T00:00:00","fromExcluded":false,"toExcluded":true},"dsp":"2015-01-
01T00:00:00Z"},"has-more-rel":false,"rel":[]},{"field":"Last
Modified","optional":"none","match":"exact","exclude":false,"count":53638,"weight":
0,"facts":{},"grp":-1,"val":{"act":{"type":"range","from":"2016-01-
01T00:00:00","to":"2017-01-
01T08:00:00","fromExcluded":false,"toExcluded":true},"dsp":"2016-01-
01T00:00:00Z"},"has-more-rel":false,"rel":[]}],"facts":{},"params":
{},"display":"Last Modified","max-count":156357.0,"min-count":0.0,"sum-of-
counts":338193.0}, }, results: [] }; ResponseService.setResponse('response',
$scope.response); }]);
```


Result



widget:spelling-suggestions

Description

Usage

as element:

```
<widget:spelling-suggestions response="{string}" query="{string}">
</widget:spelling-suggestions>
```

Parameters

Param	Type	Details
response	**string**	The name of the response from which to retrieve the spelling suggestion messages.
query	**string**	The name of the query originally submitted by the user. This can differ from the one on the Response which was sent to the search platform.

Example

Source

```
<h3 id="example_not-auto-corrected">Not Auto-Corrected</h3> <h4 id="example_not-
auto-corrected_default">Default</h4> <tk-example> <widget:spelling-suggestions
response="notAutoCorrected" query="query"></widget:spelling-suggestions> </tk-
example> <tk-code-wrapper> <code hljs hljs-no-escape>&lt;widget:spelling-
suggestions response="notAutoCorrected" query="query"&gt;&lt;/widget:spelling-
suggestions&gt;</code> </tk-code-wrapper> <hr /> <h4 id="example_not-auto-
corrected_with-label-and-suffix">With Label and Suffix</h4> <tk-example>
<widget:spelling-suggestions response="notAutoCorrected" query="query" label="Were
you looking for:" suffix="?"></widget:spelling-suggestions> </tk-example> <tk-code-
wrapper> <code hljs hljs-no-escape>&lt;widget:spelling-suggestions
response="notAutoCorrected" query="query" label="Were you looking for:"
suffix="?"&gt;&lt;/widget:spelling-suggestions&gt;</code> </tk-code-wrapper> <hr />
<h3 id="example_auto-corrected">Auto-Corrected</h3> <h4 id="example_auto-
corrected_default">Default</h4> <tk-example> <widget:spelling-suggestions
response="autoCorrected" query="query"></widget:spelling-suggestions> </tk-example>
<tk-code-wrapper> <code hljs hljs-no-escape>&lt;widget:spelling-suggestions
response="autoCorrected" query="query"&gt;&lt;/widget:spelling-suggestions&gt;
</code> </tk-code-wrapper> <hr /> <h4 id="example_auto-corrected_with-label-and-
suffix">With Label and Suffix</h4> <tk-example> <widget:spelling-suggestions
response="autoCorrected" query="query" original-label="Couldn't find anything for:"
original-suffix=". :(" corrected-label="How about:" corrected-suffix="? :)">
</widget:spelling-suggestions> </tk-example> <tk-code-wrapper> <code hljs hljs-no-
escape>&lt;widget:spelling-suggestions response="autoCorrected" query="query"
original-label="Couldn't find anything for:" original-suffix=". :(" corrected-
label="How about:" corrected-suffix="? :)"&gt;&lt;/widget:spelling-suggestions&gt;
</code> </tk-code-wrapper> <hr />
```

```
angular.module('lightning') .controller('ExampleController',
['$scope','$timeout','ResponseService','QueryService',
function($scope,$timeout,ResponseService,QueryService) { $scope.notAutoCorrected =
{ page: 2, query: { rpp: 50 }, facets: {}, results: [], "messages": [ { "type":
"spelling", "count": 0, "label": "Twigkit", "query": "Twigkit", "auto-corrected":
false } ] }; $scope.autoCorrected = { page: 2, query: { rpp: 50 }, facets: {},
results: [], "messages": [ { "type": "spelling", "count": 0, "label": "Twigkit",
"query": "Twigkit", "auto-corrected": true } ] }; $scope.query = { q:'tiwkit' };
ResponseService.setResponse('notAutoCorrected', $scope.notAutoCorrected);
ResponseService.setResponse('autoCorrected', $scope.autoCorrected);
QueryService.setQuery('query',$scope.query); }]);
```

Result

Not Auto-Corrected

Default

EXAMPLE

Did you mean: Twigkit

```
<widget:spelling-suggestions response="notAutoCorrected" query="query"></widget:spelling-
suggestions>
```

With Label and Suffix

EXAMPLE

Were you looking for: Twigkit?

```
<widget:spelling-suggestions response="notAutoCorrected" query="query" label="Were you looking
for:" suffix="?"></widget:spelling-suggestions>
```

Auto-Corrected

Default

EXAMPLE

No results found for tiwkit. Showing results for Twigkit.

```
<widget:spelling-suggestions response="autoCorrected" query="query"></widget:spelling-
suggestions>
```

With Label and Suffix

EXAMPLE

Couldn't find anything for: tiwkit. :(How about: Twigkit? :)

```
<widget:spelling-suggestions response="autoCorrected" query="query" original-label="Couldn't
find anything for:" original-suffix=". :(" corrected-label="How about:" corrected-suffix="?
:)"></widget:spelling-suggestions>
```

widget:spinner

Description

Creates a loading spinner

Usage

as element:

```
<widget:spinner> </widget:spinner>
```

Example

Source

```
<widget:spinner></widget:spinner>
```

Result

widget:tutorial

Description

Checks a cookie to determine whether to show the content and sets the cookie when an event is triggered

Usage

as element:

```
<widget:tutorial cookie="{string}" [event="{string}"] [delete="{string}"]>
</widget:tutorial>
```

Directive info

- This directive creates new scope.

Parameters

Param	Type	Details
cookie	**string**	The name of the cookie to check and set
event *(optional)*	**string**	The name of the event to set the cookie. Defaults to cookie attribute '_set'
delete *(optional)*	**string**	The name of the event to delete the cookie. Defaults to cookie attribute `'_delete'`

App Studio Concepts

The concepts section contains articles that are explanatory in nature. It provides a high-level description the various components, functions, and aspects of App Studio.

To get started, see *Downloads*.

For reference information, see the *reference section*.

- ***App Development***
  App Studio supports customization using an array of markup elements, or tags, that handle all the complexities of connecting to Fusion and visualizing the response to your queries.
- ***App Studio Enterprise***
  App Studio Enterprise is a template for building a standalone Appkit application for Fusion. All of the instructions are self contained in the README and relevant configuration files and will lead you through the simple and easy process of configuring an App Studio application.
- ***Configuration***
  The articles in this section describe how to configure a search application.
- ***Extending Appkit***
  The Appkit application framework adheres to standard Java web application methodologies making custom development using generally accepted principles perfectly acceptable. In addition to that Appkit provides a number integration hooks for any aspect of the lifecycle and extension points for adding custom capabilities and features to Appkit application.
- ***Layout and Styling***
  The articles in this section describe the basics of layouts and styling.
- ***Navigation***
  The articles in this section describe Appkit tags used for navigation.
- ***Overview***
  The Appkit library of search application components has simple building blocks that can be put together to build complex applications for search and information retrieval.
- ***Rendering Search Results***
  Search results are the focus of every search interface, and Appkit makes it easy to forge the results precisely to your liking.

App Development

App Studio supports customization using an array of markup elements, or tags, that handle all the complexities of connecting to Fusion and visualizing the response to your queries.

These topics explain the details of how to customize your search interface:

- *Introduction to App Studio Markup*
- *Anatomy of A View*
- *Display Search Results*
- *Style an App*
- *Add Navigation Tools*
- *Search Queries*
- *Search Filters*
- *Configuring Autocomplete*
- *Localize an App*
- *Security*
- *Connecting to other search providers*
- *Demo Deployment*

You can edit the default view in your browser using the *code editor* or you can *download the project* in order to develop it using your own IDE or store it in a source control repository.

- You can get detailed information about every App Studio markup tag in the *Reference Guides*.
- For ideas and reusable code, see the *App Studio Gallery*.

App Studio Enterprise

App Studio Enterprise is a template for building a standalone Appkit application for Fusion. All of the instructions are self contained in the README and relevant configuration files and will lead you through the simple and easy process of configuring an App Studio application.

To get started, *download the latest copy*, and check out the instructions *here*.

> Starting in Fusion 5.0, App Studio is no longer included in the Fusion UI. However, you can use App Studio Enterprise (ASE) to create apps in Fusion 5 *and* Fusion 4. See *How to Deploy App Studio Enterprise* for instructions.

App Studio Enterprise Downloads

Version	Compatible with	Download link
App Studio Enterprise 4.14.0	Fusion 4.x, 5.x	Zip *(https://storage.googleapis.com/docs-downloads/app-studio-enterprise-4.14.0.zip)* SHA-256 encrypted zip *(https://storage.googleapis.com/docs-downloads/app-studio-enterprise-4.14.0.zip.sha256)*
App Studio Enterprise 4.13.0	Fusion 4.x, 5.x	Zip *(https://storage.googleapis.com/docs-downloads/app-studio-enterprise-4.13.0.zip)* SHA-256 encrypted zip *(https://storage.googleapis.com/docs-downloads/app-studio-enterprise-4.13.0.zip.sha256)*
App Studio Enterprise 4.11.0	Fusion 4.x, 5.x	Zip *(https://download.lucidworks.com/app-studio-enterprise-4.11.0/app-studio-enterprise-4.11.0.zip)* SHA-256 encrypted zip *(https://download.lucidworks.com/app-studio-enterprise-4.11.0/app-studio-enterprise-4.11.0.zip.sha256)*
App Studio Enterprise 4.10.0	Fusion 4.x, 5.x	Zip *(http://download.lucidworks.com/app-studio-enterprise-4.10.0/app-studio-enterprise-4.10.0.zip)* SHA-256 encrypted zip *(http://download.lucidworks.com/app-studio-enterprise-4.10.0/app-studio-enterprise-4.10.0.zip.sha256)*
App Studio Enterprise 4.9.0	Fusion 4.x, 5.x	Zip *(https://download.lucidworks.com/app-studio-enterprise-4.9.0/app-studio-enterprise-4.9.0.zip)* SHA-256 encrypted zip *(https://download.lucidworks.com/app-studio-enterprise-4.9.0/app-studio-enterprise-4.9.0.zip.sha256)*
App Studio Enterprise 4.8.0	Fusion 4.x, 5.x	Zip *(https://download.lucidworks.com/app-studio-enterprise-4.8.0/app-studio-enterprise-4.8.0.zip)* SHA-256 encrypted zip *(https://download.lucidworks.com/app-studio-enterprise-4.8.0/app-studio-enterprise-4.8.0.zip.sha256)*

Get Started with ASE

With App Studio Enterprise you can build a mobile-ready enterprise search application in a matter of minutes. Follow the steps below to get up and running!

Development tools

In building out a search application, we recommend the use of the following tools:

- Java 1.8 JDK
- NodeJS & npm for front-end dependency management
- An IDE (Intellij, VS Code, Eclipse..)

Connecting to Fusion

We just need to edit a few config files - fire up your favorite text editor! All of the config files can be found within `src/main/resources/conf`.

Edit `services/api/fusion.conf` - this file controls which Fusion to use:

- Set the value of `host` to be the hostname of your Fusion, e.g. `host: localhost` or `host: fusion.example.com`
- Set the value of `port` to be the port of your Fusion - this is `6764` by default, which is preconfigured for you.

Edit `platforms/fusion/data.conf` - this file configures which data in Fusion to use:

- Replace the text `YOUR_QUERY_PROFILE` with the name of the Fusion Query Profile you wish to direct your queries through. If you have created an app via the Fusion UI called `MyAwesomeApp`, then you will have a default Query Profile called `MyAwesomeApp` as well - this is a good starting point: e.g. `query-profile: MyAwesomeApp`.

Edit `platforms/fusion/social.conf` - this file configures the Lucidworks Appkit Social Module:

- Replace the text `YOUR_FUSION_APP` with the name of your Fusion app, as used above: e.g. `collection: YOUR_FUSION_APP_user_prefs` becomes `collection: MyAwesomeApp_user_prefs`. This configures the Appkit SDK to store social data in the `COLLECTION_NAME_user_prefs` sidecar collection associated with `MyAwesomeApp`.

Edit `platforms/fusion/suggestions.conf` - this file configures search suggestions:

- Replace the text `YOUR_QUERY_PROFILE` with the name of the Fusion Query Profile you wish to direct your search suggestion queries through. This can be the same Query Profile as you are using for search queries, e.g. `query-profile: MyAwesomeApp`.

Edit `message/service/fusion.conf` - this file configures how Signals are sent to Fusion:

- Replace the text `YOUR_QUERY_PROFILE` with the name of your query profile, as used above: e.g. `query-profile: MyAwesomeApp`.

Starting and Stopping

Linux, Mac OS X

To start the application, run the following command from a terminal:

- `./app-studio start`

To stop the application, run the following command from a terminal:

- `./app-studio stop`

To package the application into a WAR file, run the following command from a terminal:

- `./app-studio package`

Windows

To start the application, run the following command from a command prompt:

- `app-studio.bat start`

To stop the application, run the following command from a Powershell command prompt:

- `app-studio.bat stop`

To package the application into a WAR file, run the following command from a command prompt:

- `app-studio.bat package`

See *Additional script commands* for more commands.

Accessing App Studio

Once started, the application is served up at `http://localhost:8080`. You will be asked to authenticate, for which you can use any of your Fusion "native realm" users - if you are just getting started with Fusion, this will be your Fusion `admin` username and password.

When building an application for production environments the security realm can be configured in the `src/main/resources/conf/security/fusion.conf` file.

Using Your Data

The file `src/main/webapp/views/search.html` defines the main page layout of your application. App Studio Enterprise is pre-configured to work out of the box with Lucidworks Fusion, but you may wish to customize which fields you use in your results.

Let us get started!

Result title

The title of each result document can be configured like so:

Find the tag that looks like this:

```
<search:field name="title_s" styling="title" urlfield="id"></search:field>
```

- `styling="title"` means that it will be used as the result title - no need to change this, but it will help you find the right tag!
- `name="title_s"` configures the application to use the value of the `title_s` field as the displayed title text for each result. If your document titles are stored in a different field, replace `title_s` with the name of that field.
- `urlfield="id"` configures the application to use the value of the `id` field as the hyperlink - when users click on the title, they will be navigated to whatever URL is in the `id` field for that result. If your document URLs are stored in a different field, replace `id` with the name of that field.

Result description

The text description of each result document can be configured like so:

Find the tag that looks like this:

```
<search:field name="description_s" label="Description" max-characters="240" show-
more></search:field>
```

- `label="Description"` will help you find the right tag.
- `name="description_s"` configures the application to use the value of the `description_s` field for the description text of each search result. If your descriptions are stored in a different field, replace `id` with the name of that field.

Facets

Facets are a handy way to refine your search. By default, App Studio Enterprise will attempt to show facets for the `keywords_s` field, if it exists. We can add or remove facets very simply:

Find the tag that looks like this:

```
<search:query var="query" parameters="*" results-per-page="12" facets="keywords_s">
</search:query>
```

This tag creates the overall page query - `facets="keywords_s"` adds the ability to facet on the `keywords_s` field to the query. We can remove this, or add some more facets - this parameter takes a comma-separated list of field names.

Try `facets="keywords_s,charSet_s"`, or replace the value entirely with a list of your own facet fields.

Activating your changes

To see your changes in action: - Save the file - Refresh your browser - Yes, that is all!

Development

Resources

App Studio Enterprise is preconfigured to automatically reload when you make changes to your code during development.

The following file types are monitored and re-processed when changed: - HTML (`.html`) - JavaScript (`.js`) - LESS (`.less`) - CSS (`.css`) - JSP (`.jsp`) - Config (`*.conf`)

There will be a slight pause between saving the file and being able to view the changes as App Studio Enterprise processes the file. Refresh your browser after about 2 seconds to see the changes.

Java

Hot reloading of Java class files requires some extra steps. The Java Virtual Machine supports hot-swapping of class files as long as the changes are not structural - changes to method bodies and field values are supported, but adding or renaming methods are not. You will need to use an IDE to set this up - the following steps assume you are using JetBrains IntelliJ IDEA:

1. Import the project into IntelliJ
2. Add a new Maven "Run Configuration" - from the "Run" menu, select "Edit Configurations". Click the "+" button, and select "Maven" from the dropdown.
3. Enter a name for your configuration - e.g. "Jetty Run"
4. Enter the following for the "Command Line": `clean jetty:run -Pdevelopment --settings ./bin/settings.xml -Dtwigkit.conf.watch=true`, then click "Ok".
5. Make sure that you are not already running App Studio Enterprise.
6. From the "Run" menu, click "Debug 'Jetty Run'". If you have named your configuration something other than 'Jetty Run', look for "Debug '<your configuration name here>'"
7. As App Studio Enterprise starts, you will see the console output within your IDE

We are now set up to hot reload Java files:

1. Edit a Java file
2. Save the file
3. From the "Build" menu, select "Recompile <your class name>.java"
4. You will see a popup notifying you that a class file has changed, and asking if you would like to reload classes - click "Yes".
5. Your Java change has been deployed into the running Java Virtual Machine - you can now test your running App Studio Enterprise webapp and see your change in action.

License

App Studio Enterprise comes with a 30-day trial license in `app-studio.lic`. Once this expires, contact Lucidworks for a new license: https://lucidworks.com/contact/ *(https://lucidworks.com/contact/)*

Additional script commands

	Each command and flag is also applicable to the batch script on windows. === start

```
./app-studio start [-f] [-p port] [-P profile] [-m memory] [-t timeout] [-V]
```

`-f` Start App Studio in foreground.

`-p <port>` Specify the port to start the App Studio web server on; default is 8080.

`-P <profile>` Specify the Appkit profile to use when App Studio starts; defaults to development.

`-m <memory>` Sets the min (-Xms) and max (-Xmx) heap size for the JVM, such as: -m 4g results in: -Xms4g -Xmx4g; by default, this script sets the heap size to 512m.

`-t <timeout>` Sets the startup timeout in seconds (defaults to 240).

`-V` Verbose messages from the startup script.

stop

```
./app-studio stop [-p port] [-V]
```

`-p <port>` Specify the port the App Studio HTTP listener is bound to.

package

```
./app-studio package [-P profile]
```

`-P <profile>` Specify the Appkit profile to use when App Studio is packaged; defaults to development.

This command will build a new JAR and WAR file with App Studio and place it in `./dist`.

docker

```
./app-studio docker [package] [start] [-P profile] [-i image_name]
```

`package` Package App Studio Enterprise then build a Docker image.

`start` Package App Studio Enterprise, build a Docker image, then run a container on port 8080.

`-P <profile>` Specify the Appkit profile to use when App Studio Enterprise is packaged; defaults to development.

`-i <image_name>` Specify the image name to use when App Studio Enterprise is built as a Docker image; defaults to app-studio-enterprise. The image will be tagged as the latest.

Advanced Configuration

Add Custom CSS

Add **`class`** to any component that needs to be customized in an **`.html`** file. for example: **`customClass`** has been added.

```
<search:field name="field_name" label="label" styling="description"
class="customClass"></search:field>
```

Then add the custom class in **`custom.less`** which is located in **`src/main/webapp/styles/includes/custom.less`**.

```
.customClass { font-weight: bold; font-style: normal; }
```

Add Custom JavaScript

App Studio is configured out of the box to allow you to simply add your JavaScript inside the controller function in **search.controller.js** and have it trigger and function on every page in your app.

```
// Print the current page to the console $scope.currentPage = window.location.href;
console.log($scope.currentPage);
```

Running the app over SSL

The webapp can also be served over HTTPS using SSL encryption. This is done using the following parameters when invoking the startup scripts:

- `-Dtwigkit.ssl=true` required to turn on secure mode

We include a keystore file with a default self-signed key for development / testing of secure mode. However in order to establish proper trust you will need to import your own valid SSL certificate into the file `keystore.jks` or create your own keystore file. You can then configure SSL via:

- `-Dtwigkit.https.port` optionally sets the HTTPS port (defaults to 8765)
- `-Dtwigkit.http.port` optionally sets the HTTP port (defaults to 8080)
- `-Dtwigkit.keystore.file` keystore filename / location relative to webapp (defaults to 'keystore.jks')
- `-Dtwigkit.keystore.password` password to the keystore (defaults to 'p4ssw0rd')
- `-Dtwigkit.keystore.alias` name of the key in the keystore to be used (defaults to 'default-key')

Twigcrypt value encryption utility

A best practice that we advocate is to encrypt any sensitive configuration values using our Twigcrypt utility. Twigcrypt's function is to allow you to quickly encrypt sensitive string values using the command line. These files can be found at `/bin/twigcrypt.zip`.

For more details on how to use Twigcrypt, refer to the README inside the zip file.

Appkit

The Appkit library of search application components has simple building blocks that can be put together to build complex applications for search and information retrieval.

Appkit provides a generic API for searching and indexing content that completely abstracts the search engine or database in use. This means the application developer does not need to know the ins and outs of each search and database technology, but instead only needs to know how to work with Appkit.

At the most fundamental level:

1. Appkit sends generic queries (which are platform independent) to a platform that translates them into something the underlying search engine can work with.
2. Appkit maps the response from the search engine to a generic Appkit search response which other components can work with and that Appkit UI widgets can render.

Appkit Concepts

Platform

Appkit integrates with a variety of *search platforms*. These are the back-end technologies responsible for indexing the documents for efficient retrieval of search results. These are often referred to as 'search engines' or 'search technologies', but throughout the Appkit framework, we standardize on the term 'platform'.

A platform provides the means to query a collection of data, returning a list of results. The original format of the data could be full-text documents, media (audio/video/images), or structured relational data.

In our documentation, we assume that the target search platforms for the application you are developing are configured and ready to query.

Query

A query is built up from the variety of search controls available in the application to form a bundle of parameters that the platform will understand. Appkit will automatically convert the query into the platform's native format. It will then execute the query against the platform and return the results matching the query.

Queries vary in complexity from a search on a single term to combinations of many 'clauses' with boolean logic (AND/OR/NOT). As far as a user is concerned the search controls provided in Appkit make it easy to build queries which best represent their information need by following intuitive and well established user experience (UX) patterns.

Response

A search response wraps all the information a platform returns from a query including the result list and facets.

Result list

A result list is the collection of data matching a query issued against a platform.

Result

Each result in a result list will contain information stored in named fields. This information can take any form. Appkit provides tags for displaying the fields within a result including styling for displaying common field types. Every result has an `id` field that allows it to be uniquely identified amongst others in the collection of data being searched.

Facets

Facets can be thought of as groupings or categories of results. Typically a platform will return all the values of facets within the results along with a count of the number of results within that category matching the given query.

The common terminology for returning facet values for a particular field is '*faceting on* a field'. Platforms handle facets differently and the facets to return can usually either be statically configured to return with all search results, or dynamically specified with each query sent to the platform. The sensible defaults in Appkit let you easily display all available facets without prior knowledge of those which are statically configured.

Facets allow for a richer search experience as users can click on a particular facet value to limit the results to those containing only the selected value of the field which is faceted on.

Pagination

Nearly all search pages have a limit to the number of results displayed on a page. Pagination refers to the ability to navigate between the pages of results. This can vary in implementation and context. For example when viewing results on a desktop or tablet, you might be given a list of page numbers to choose from, whereas the same page on a mobile device might display only Next and Previous links. Pagination is usually combined with a control that lets the user choose between some options for the number of search results which appear on a page (for example, 5, 10, 50, 100).

Sort order

The order in which results appear is dependent on how the platform is configured and the 'sort' options applied (if any) in the query. The default *sort order* for most platforms organizes the results by relevance to the query. This is usually based on a calculation using a number of factors. Full-text relevancy factors include: the number of times terms in the query appear in the document, the rarity of the terms across the collection of documents, the number of clauses in a query which are matched by the document and normalization factors such as the length of a field in which a match is found.

Usually in a search application some options for re-sorting a result list based on a particular field are provided (for example, publication date, customer rating).

Relevancy ranking (score)

Most platforms sort results using a calculation of the relevance of a document in the result list to the query. This is often returned as a number in a 'score' field within each result representing the ranking of a document relative to others in the result list. Scores are usually fairly meaningless on their own and only make sense when compared to other results in the same result list.

Filters

Filter is a general term for anything that limits the search results. Many of the Appkit controls allow the user to apply filters to narrow down the results to those they seek. For specific types of fields, there are different options for how a filter can be represented. For example date fields can be filtered on using date pickers to choose a range or plain text to represent distinct ranges available to filter on.

Most field values represented in an Appkit application are exposed as links which can be clicked on to allow a user to add a filter for that specific value. An example of this is that every slice of a pie chart visualization of the results is a clickable link which will filter to only those results which match the value represented by the slice of the pie chart.

Careful thought to filters and how best to represent them in a search application can improve the quality of the search experience and efficiency of the search.

Appkit tags

Appkit provides a range of client-side tags for building and executing search queries as well as displaying, formatting and visualization of search results.

These tags are packaged as AngularJS libraries divided into a number of namespaces. See the *Tag Reference* for full documentation on each of the tags.

Most Appkit tags can be used in various combinations and are designed to work together within the framework to deliver a high quality search experience for the user.

Common usage

This example demonstrates how the result-list tag works in line with our principles of *configuration over code* and *sensible defaults*.

To render a simple results list displaying every field in each document, this code is all that is required in the JSP:

```
<search:result-list response="response"></search:result-list>
```

If you want to go ahead and customize the output of a result list, you can easily do so along these lines:

```
<search:result-list response="response"> <search:result> <search:field name="title"
style="title"></search:field> <search:field name="url" style="url" label="Link:">
</search:field> <search:field name="description" style="description"
label="Abstract:"></search:field> </search:result> </search:result-list>
```

This demonstrates a common pattern in Appkit of basic tags having default rendering to get you started, with the ability to customize the rendering of elements within them to a suitable level of granularity.

Configuration

Many tags have reasonable default configurations, but they can also pick up attributes from configuration files, avoiding the must change any JSP code:

Here is a platform tag which looks for properties in the configuration file at `src/main/resources/conf/platforms/myplatform/dev.conf`

```
<search:platform var="platform" conf="platform.myplatform.dev"></search:platform>
```

See section on *configuration files* for more information.

Architecture and Technology Stack

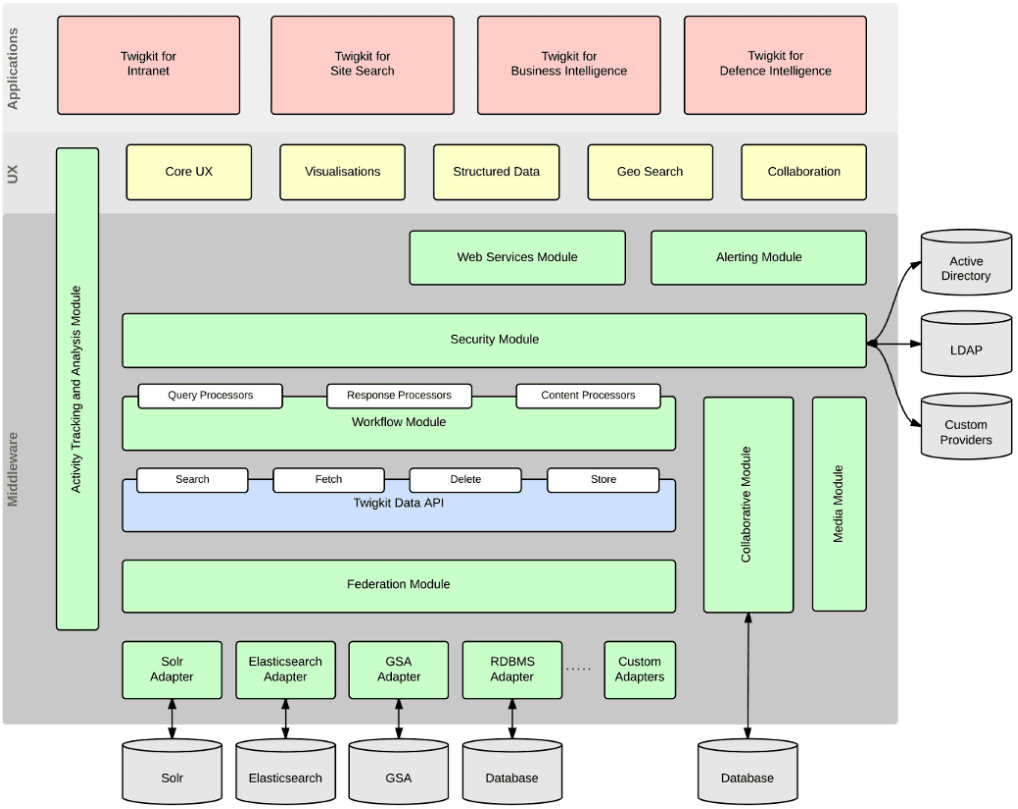
Appkit is a stateless, modular, 100% Java Standard Edition software framework.

Architecture

Appkit consists of:

- A layer of *data-abstraction middleware*
- A collection of functional *modules* for data manipulation
- *User interface components* for rendering and visualizing the data

An Appkit search application consists of modular building blocks, which are pieced together through the use of special *Appkit tags*, which can be mixed in with standard HTML and JSP code.



Technology stack

The Appkit stack is designed to be lightweight from the point of view of efficiency and ease of development. No programming or Java code is necessary to build a search application with Appkit.

Software that Appkit uses

Appkit uses:

- Maven for server side dependencies and build management
- npm for client side dependencies
- Standard Java Servlets and JSPs
- Standards-compliant AngularJS, HTML, CSS, and JavaScript

What Appkit provides

Appkit provides:

- A self-service application generator for project generation
- AngularJS based tag libraries for all framework components

- Web services to expose all search functionality
- Extension points for custom tags, JavaScript, and CSS

Deployment options

Appkit applications can be:

- Deployed locally in a development environment using the Maven Jetty plugin
- Built to a standard WAR file for deployment on Tomcat (or any other servlet container/application server)

Data Sources

Appkit can obtain and present information from multiple sources simultaneously, and seamlessly integrate with all common search engines, data warehouses, web service endpoints. The platform abstraction covers both the storage and sides, meaning that Appkit implements for each platform the necessary protocols to save and index content. A number of platform adapters are available out-of-the-box with Appkit, and *adding new ones* is a straightforward exercise.

Appkit is not limited to presenting information from search engines. It also supports data warehouses, web service endpoints, and other information sources that expose an API.

Platform adapters

The layer that relays Appkit specific instructions to the underlying data provider is referred to as as Platform Adapters. This is responsible for translating for example, an Appkit query to a corresponding command the underlying platform will understand; and conversely an engine specific response to a generic Appkit response.

The platform abstraction covers both search and storage sides, meaning that Appkit will implement for each platform the necessary protocols to save or index content for any platform.

A number of platform adapters are available out-of-the-box with Appkit, and adding new ones is a fairly trivial exercise.

Search engines

- *Fusion*
- *Solr*
- *SolrCloud*
- *Elasticsearch*

Other data sources

- *Wolfram|Alpha*
- *Wikipedia*
- *Pipl*
- *YouTube*
- *Twitter*
- *Google Custom Search*

Store, search, load, delete

Appkit (assuming storage backend capabilities) provides support for storing or indexing, deleting, fetching individual records, and searching the backing data store.

Task support

Appkit supports long-running, managed objects called *Tasks*. This is useful for operations that can take longer to complete such as Map/Reduce processes. In that case Appkit will manage the lifecycle, initiate the task, track and report progress, and finally present the response. After a response is received, it can be rendered using Appkit user interface components, or even passed to another platform for persistence which might give you low-latency searching and faceting.

Batch operation support

Appkit can cursor and retrieve entire indexes by combining Task Support and our Batch Platform Wrapper which will automatically offset through any number of results from a platform and combine the responses.

Routing and URL Rules

This page explains the general URL patterns used in the application to map URI paths to JSP pages and resources within an Appkit application.

The URL routing is handled by the Tuckey URL rewrite filter *(http://tuckey.org/urlrewrite/)*, and configured using this file: `src/main/webapp/WEB-INF/url-rules.xml`

This filter allows particular URLs to be served by specific JSPs, without forwarding or redirecting the user, thus the URL will remain the same in their browser. It provides a flexible way to define rules for mapping URLs in the application using regex *(http://en.wikipedia.org/wiki/Regular_expression)*.

By default everything under the application context is passed through the filter, however most Appkit applications define a pattern which are passed straight through unmodified (essentially by-passing the filter):

```
<rule> <name>Ignore file, non-jsp and non-query requests</name> <condition
type="request-uri" next="or">^/twigkit/.*</condition> <from>.*</from> <to
last="true">$0</to> </rule>
```

The remaining requests for URLs not matching the patterns to be ignored (shown above) will be processed through rules designed for the bulk of user requests such as search queries. A typical configuration for an Appkit application is shown below:

```
<rule> <from>^/login/</from> <to last="true">/login.jsp</to> </rule> <rule>
<from>^/login</from> <to last="true">/login.jsp</to> </rule> <rule match-
type="wildcard"> <name>Ignore files on URL path.</name> <from>/**.*</from> <to
last="true">/$0</to> </rule> <rule match-type="wildcard"> <name>HTML 5 URL
Rewrite</name> <from>/**</from> <to last="true">/index.jsp</to> </rule>
```

The first two rules routes all requests for '/login' or '/login/' to the login page. This will typically be the user's entry point to the application, though if security is enabled and the user is unauthenticated, any other page name given will direct the user to the login page by default.

The third rule matches all requests that have a period in the path (for example, 'mypage.html') and breaks the request.

The fourth rule will route all remaining requests to the index.jsp page, whereby the angular code will interpret the URL via the angular routing mechanism, and point to the correct view.

Each of these rules can be modified to support different patterns, depending on the requirements of the application. By default all GET request parameters, such as Appkit's *q* parameter representing the search query (**?q=my+search+terms**), will be passed straight through to the page that the request is routed to. Before modifying these rules, we recommend that you read the URL rewrite filter documentation *(https://cdn.rawgit.com/paultuckey/urlrewritefilter/master/src/doc/manual/4.0/index.html)*, and specifically the section about the `urlrewrite.xml` configuration file *(https://cdn.rawgit.com/paultuckey/urlrewritefilter/master/src/doc/manual/4.0/index.html#configuration)*, before modifying these rules.

Signals in Appkit

Appkit provides support for the *Fusion REST Signals API*, allowing user clicks and the creation of annotations (bookmarks, topics etc) to be recorded and tracked.

To learn how to use these signals with Appkit, see *Incorporating Fusion Signals in Appkit Applications*.

Creating a collection for signals tracking

Using the *Fusion UI*, when a primary data collection is created, this also creates a collection for the signals. By default, the name of this signals collection is by `<primary collection>_signals`, where `<primary collection>` should be substituted with the name of the primary data collection. After data has been indexed and ingested into the primary collection, the Appkit Fusion Signals module can be used to populate the signals collection.

Tracking user clicks and annotations

Currently, Appkit will send information on user clicks that trigger URL requests as well as when a user creates an annotation, for example, a bookmark or comment.

A signal event of type `click` will include, amongst other things, information about the URL that was clicked as well as the number of times that URL was clicked.

A signal event of type `annotation` will include, amongst other things, information about the annotation target, the collection, and the creator.

For a full list of properties that are stored, refer to the signals collection associated with your application.

Authentication

Activity tracking requires authentication. Both *service account impersonation* and the Fusion *Sessions API* can be used for this purpose.

Service account impersonation requires credentials for the service account and these are stored in the Fusion platform configuration. These credentials can be accessed by setting the name of the Fusion platform configuration via the `platform-config` parameter in the Fusion activity tracking configuration.

If the credentials for a service account are not provided or cannot be found, a check will be made to see if a Fusion session cookie is available. Typically, this will be present when the user is known to the Fusion and the Session API has been set up accordingly.

Social and Collaboration Tools

The social and collaboration tools provided by the Social Module extend the search application with the ability to store user-generated information on individual documents and document collections, and to share this information among groups of authenticated users. For example, documents can be tagged or bookmarked, and users may save search queries and add comments to individual documents.

User-generated content can be stored in a relational database or any other kind of data store, using our pluggable architecture.

Authentication and authorization of users is managed via the Security module.

Terminology

While much of the language used to describe collaborative tools is carefully chosen to be self explanatory, there are several general concepts which are used throughout and are introduced in this section.

Annotations

Key to the collaboration enabled by inclusion of this module, there are several types of *annotation* which can be applied to documents. These include bookmarks, comments, and likes, which are commonly found in many applications and provide a versatile way for users to contribute and save information during their search.

Topics

A *topic* is a general term for a subject that user(s) are intending to collect information about. This module provides facilities for users to create a named group with which they can associate any number of annotated documents. Topics themselves can be created and selected using the *topic form tag*. Many of the collaborative tags included in this module have a *topic* attribute which is used to link the saved information to a specific topic (which can be picked up from the URL - more on this in the Topic pages module.

Topics can be defined by the developer as being 'namespaced' so they can be global (available for all users to collaborate on) or local (only viewable by the individual user that created them).

Collections

Some annotations can be categorized by *collection*. This is a mechanism for separating annotations of different types of document. There is a subtle difference between topics and collections, in that collections are usually determined by the categories of information in the data and generally pre-defined by the application developer offering a small number of collections with which to associate annotations. Collections are intended for a finite number of categorical groups whereas any number of topics can be created.

Combining social and collaboration tools

Topics and collections can even be combined such that a user can create a named 'bucket' to group information together, but have the saved information partitioned. One example is enabling the user to construct a wishlist of items that are automatically separated by category.

User Interface Components

Appkit has distilled years of research, design, and usability testing into a library of configurable user interface components responsively designed for both desktop and mobile.

These include support for:

- **Faceted navigation.** All common interaction modes, including text links and checkboxes, lists and hierarchies.
- **Advanced search.** Advanced search form components that utilize various matching strategies for deeper searching.
- **Suggestions.** As-you-type suggestions driven by a static list of terms, based on a field in the index, or powered by the platform.
- **Data visualizations.** From simple bar charts, to pie charts, to sophisticated area and line graphs for search-driven BI.
- **Geospatial mapping.** Plot search results on interactive maps from any provider, such as Google or Bing.
- **Search controls.** All the essentials you would expect: sorting, pagination, results per page, breadcrumbs, tabs.
- **Topic pages.** Search-driven topic pages using human, readable, "pretty" URLs, such as: mywebsite.com/companies/amazon/.

The Appkit framework removes the need for low-level code when developing search applications. Everything from establishing a connection to the search platform through to rendering results and facets can be granularly controlled via simple markup alongside regular HTML. Applications built using Appkit are **mobile-optimized** through responsive design and will correctly adapt to tablet and mobile sized devices.

In general, there is no must create a separate site for a mobile view, although that is also supported. Appkit provides an optimized experience for modern browsers (using HTML5 document types) but degrades gracefully and is functional back to IE11.

Applications can be fully **localized** (Java I18N), and our media module lets you normalize image size and resolution, on-the-fly in an efficient (cacheable) manner.

Configuration

The articles in this section describe how to configure a search application.

- *Basic Configuration*
  Appkit gives you full flexibility to customize your search application, whilst providing sensible default settings out of the box. A typical search page consists of two parts: *preparation of the data model* (preparing a query, configuring one or more search platforms, and submitting the search query), and *presentation of the search results*. Both the data model and all UI components are configured through JSP tags, which allows for rapid prototyping and agile development.
- *Configuration Files*
- *Externalising Configuration Files*
  A commonly asked question is whether you can obtain configuration files from a location external to the application, that is, somewhere else on the file system. The answer is yes.
- *Localization Keys*
- *Overlaying Configuration Values*
  To fine-tune your configuration hierarchy prior to production, Appkit has added the ability to overlay configuration hierarchies via the Jetty Maven plugin. While developing an application, you can experiment with a partial configuration and add/update only those properties that you actually must update without touching the application's main configuration. When the application starts these changes are merged with the application's main configuration hierarchy.
- *Structure of a Web Application*

Basic Configuration

Appkit gives you full flexibility to customize your search application, whilst providing sensible default settings out of the box. A typical search page consists of two parts: *preparation of the data model* (preparing a query, configuring one or more search platforms, and submitting the search query), and *presentation of the search results*. Both the data model and all UI components are configured through JSP tags, which allows for rapid prototyping and agile development.

Preparing the model

Platform tags

Platform tags are used to configure a search platform, which is the service endpoint that you query to get back search results (a document result set). To configure each search platform requires a set of parameters, such as the host name of the search engine, which facets to retrieve, etc. These can either be defined inline in the JSP page (through attributes of the platform tags) or via central *configuration files*.

Since all platforms present the same interface to the end-user, you can either configure a platform using the *generic platform tag*, or the engine specific variants which will guide you in providing all the parameters they must function.

Query tags

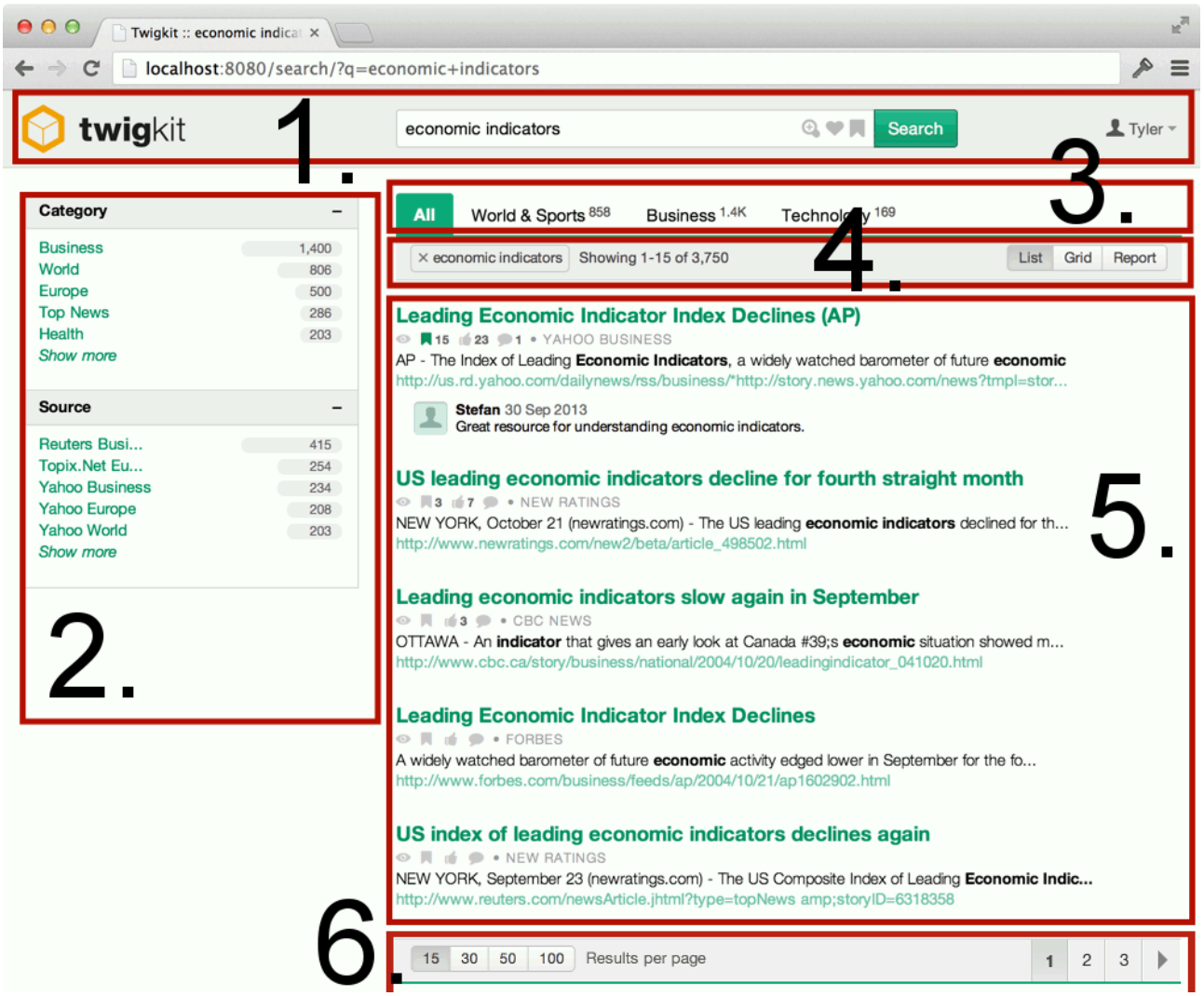
An Appkit query is made up of a number of query parameters. You can either create a query providing all the necessary parameters inline via the *Query tag* or by using query parameters submitted using HTTP request attribute. This, and the available parameters, are covered in detail on the *Query tag configuration tags page*.

Processing results on the fly

If the Workflow Module is installed, then you can use *Workflow Processors* to *modify queries* before they are submitted to the search engine, or alter the response before it is rendered.

Rendering search results

This figure shows a typical search page, with a mixture of faceted navigation, collaborative tools, and search controls. The composition of the page through atomic Appkit building blocks is outlined in detail below.



1. Search header

Most search pages will contain a header with the main search box which creates a starting point for a user to form their initial query.

Appkit search headers typically consist of:

- Logo and Branding
- Search box (with optional advanced search, bookmarks and saved search icons)
- Details of the current user including an avatar, username and user actions such as 'log out'

Associated tags:

- `<search:box>`
- `<security:user-actions>`

2. Faceted navigation

A staple of the search experience, faceted navigation allows a user to quickly refine their search, as well as giving an overview of the type of results matching their query.

Appkit provides great defaults for displaying facets, but can also be configured to allow for combining (OR'ing) filters within a category or removing all results with a particular value (adding NOT filters).

Associated tags:

- `<search:facetList />`
- `<search:facet />`
- `<layout:sidebar>`
- `<search:facet-list>`
- `<search:facet>`

3. Tabs

Tabs allow the user to navigate between divisions of the data which are typically based on separate types of data in the results.

Appkit's tab component provides an 'All' tab which is selected by default and provides a cross-search of everything. This feature can be disabled or renamed as required. Appkit also provides a count of the results matching the query next to each tab.

Associated tags:

- `<search:tabs>`

4. Breadcrumbs and controls

This horizontal bar presents information relating to the query, allowing the user to broaden their search and modify the display of results.

In an Appkit application, you will find 'breadcrumbs' allowing the user to see the search terms and filters applied and remove them to broaden the search. The number of search results shown on the page is displayed as well as the total number of matching hits. On the right side, a number of options might be provided, depending on the context of the application. These include the ability to sort the result list, change the formatting of the results, and display visualizations and charts associated with the results of the search.

Associated tags:

- `<search:breadcrumbs response="response" …>`
- `<toggle:controls labels="List`
- `<search:sort fields="date=Date" query="query" …>`
- `<search:response-statistics response="response" …>`

5. Results

This is where a representation of the search results will appear. Appkit allows a high level of customization for the presentation of results while providing plenty of options for easily displaying common data types with a small amount of code.

Any number of the fields within the data can be presented with pre-configured styling for:

- Titles
- URLs
- Descriptions
- Inline field display
- Labelled fields
- Categories
- File formats and data types
- Dates
- Document previews

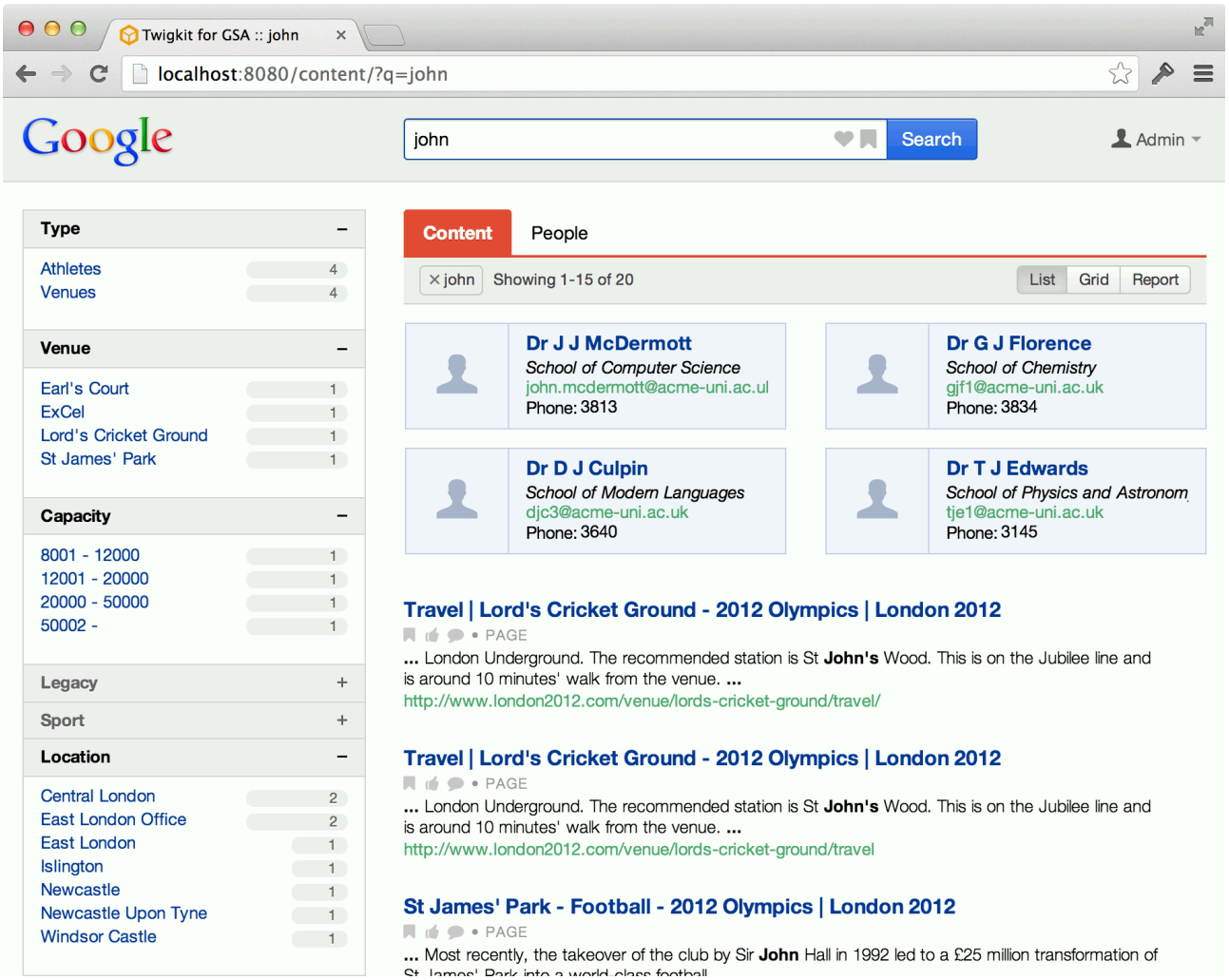
This is where links for actions such as bookmarking, likes and commenting can also be displayed.

There are a large number of tags for formatting results so we suggest referring to the documentation about *rendering search results*.

Secondary results

Often a search page will provide sections above or along-side the main results containing results from a different source. These help to give the user the option to navigate to a different type of data that might be more appropriate for the query they are trying to fulfill.

This can be implemented in a variety of ways within Appkit, see an example below of results pulled from the GSA *(http://www.google.com/enterprise/search/products/gsa.html)* platform OneBox feature:



6.Search footer

The search footer usually contains *pagination* to allow the user to move between pages of results, and occasionally a control to choose how many results to display on each page.

The tags provided allow for configuration of parameters such as the maximum number of pages to display and the options provided for the results per page.

Associated tags:

- `<search:pagination response="response" …>`
- `<search:results-per-page query="query" …>`

Configuration Files

Configuration files

You can find the configuration files in `conf/` in the code editor, or in `src/main/resources/conf/` in the project directory. The configuration files are organized by function or by module, as follows:

```
/conf /activity /message /platforms /processors /security /services
```

These configuration files support hierarchical cascading of the files via the open-source Fig *(https://github.com/twigkit/fig/)* project.

This section describes the most important configuration files in the `/config` folder:

- *config/twigkit.conf*
  Global configuration settings.
- *config/cors.conf*
  Cross-Origin Resource Sharing (CORS) configuration.
- config/platforms/fusion/fusion.conf
  Specify Fusion platform settings.
- config/platforms/fusion/data.conf
  Specify which Fusion *query profile* to use.
- config/security/fusion.conf
  Configure the Fusion security realm.

config/twigkit.conf

The App Studio application looks up global settings from a `twigkit.conf` configuration file that the application tries to locate relative to `config` on the runtime classpath.

Parameter	Type	Description
license-file	string	The path to the license file relative to the `app-studio/` directory, such as `file://./twigkit.lic`.
ui.trim-white-space	string	"True" to trim white space in the application UI.
bundleTimeToLive	long	The number of milliseconds a resource bundle (for example, a file like `languages_en.properties`), can remain in the Java resource bundle cache without being validated against the source file from which it was constructed. The value `0` indicates that a bundle must be validated each time it is retrieved from the cache. If `bundleTimeToLive` is negative, or missing, then the bundle cache will have no expiration control (i.e. entries are only evicted from cache due to runtime or memory constraints).

config/cors.conf

CORS *(http://www.w3.org/TR/cors/)* or Cross-Origin Resource Sharing is a recent W3C *(http://www.w3.org)* effort to introduce a standard mechanism for enabling cross-domain requests from web browsers to servers that wish to handle them. App Studio supports CORS filtering by default. You can control the options for the particular header attributes using the `cors.conf` file at the root of the `/conf` folder.

Parameter	Type	Description	Default
`cors.allowOrigin`	`string`	Whitespace-separated list of origins that the CORS filter must allow. Requests from origins not included here will be refused with an HTTP 403 "Forbidden" response. If set to `*` (asterisk), any origin is allowed.	`*`
`cors.supportedMethods`	`string`	List of the supported HTTP methods. These are advertized through the Access-Control-Allow-Methods *(http://www.w3.org/TR/cors/#access-control-allow-methods-response-he)* header and must also be implemented by the actual CORS web service. Requests for methods not included here will be refused by the CORS filter with an HTTP 405 "Method not allowed" response.	`GET, POST, DELETE, HEAD`
`cors.supportsCredentials`	`boolean`	Indicates whether user credentials, such as cookies, HTTP authentication or client-side certificates, are supported. The CORS filter uses this value in constructing the Access-Control-Allow-Credentials *(http://www.w3.org/TR/cors/#access-control-allow-credentials-respons)* header.	`false`
`cors.maxAge`	`integer`	Indicates how long the results of a preflight request can be cached by the web browser, in seconds. If -1 unspecified. This information is passed to the browser via the Access-Control-Max-Age *(https://www.w3.org/TR/cors/#access-control-max-age-response-header)* header.	`3600`

Hierarchical configuration

To make version control and specialization of configuration easier, the configuration is loaded hierarchically. For example, in the `/platforms` directory are subdirectories for `/fusion` and `/solr`:

```
/config /platforms /fusion /solr
```

Within the `/fusion` folder is `/fusion/fusion.conf`, which may contain the following general attributes:

```
name: twigkit.search.fusion.Fusion backwardsCompatible: true timeOut: 30000
resultIDField: id highlight: true defaultQuery: *:*
```

Within the same folder, you might find one or more configuration files which inherit or extend this one, such as `data.conf` or `people.conf`. To access a given configuration, use dot notation, such as `platforms.fusion.data` or `platforms.fusion.people`. The configuration system will traverse the hierarchy (no matter how deep) and aggregate the configuration files, overwriting attributes from higher-level files when those attributes also appear in lower-level files.

For example, if `fusion.conf` contains a `defaultQuery` attribute and you create `internal.conf` which also contains a `defaultQuery`, then the value from `internal.conf` is used. This allows you to create variations on the same platform configuration. You can then refer to any platform instance in other configuration files or in the *`search:platform` tag*:

```
<search:platform conf="platforms.fusion.internal" />
```

In this case App Studio will use the platform as configured centrally, irrespective of the search engine behind it, effectively abstracting the data provider from the view.

Configuration locations

The configuration files can be placed outside of the application and centrally accessed by multiple instances. To reference these in a different location, set the `twigkit.conf` system property to the absolute file path of the `/conf` folder, like this:

```
-Dtwigkit.conf=file://${path-to-your-conf-folder}
```

Adding special characters to key names in configuration files

To add special characters to key names in configuration files you need to escape the special characters. For example, if a key name contained a whitespace, such as `My key: value` then you would need to escape the whitespace for the configuration to be correctly loaded. In this case, the correct syntax would be:

```
My key: value
```

This rule does not just apply to whitespace but any special character that you may wish to use within a key name.

Cookies

AppKit uses three cookies: `JSESSIONID` for authentication, and `tk.query.ctx_null` and `tk.session` to track session metadata.

- `JSESSIONID` is used by Spring Security for authentication and to maintain the session for the user
- `tk.query.ctx_null` stores the current query term to track whether the query is refined
- `appkitRequestCookie` is used to prevent Cross-Site Request Forgery (CSRF) when using social features

Externalising Configuration

A commonly asked question is whether you can obtain configuration files from a location external to the application, that is, somewhere else on the file system. The answer is yes.

The inset section under **Configuration locations** *on this page* describes how to do this, but more detail is provided in this FAQ page.

You must set the `twigkit.conf` variable when running the application server (for example, Tomcat or Jetty). This is done via an extra JVM argument on startup.

For Tomcat (which is the choice for QA & PROD deployments most of our clients use), the simplest way to do this is modify the `catalina.sh` script, adding this to the end of the line starting `JAVA_OPTS=`:

```
-Dtwigkit.conf=file://{path-to-qa-or-prod-conf-directory}
```

The path in curly braces must point to the location of a valid `conf` directory structure, as found in most Appkit applications in `src/main/resources/conf`. If you have separate QA and PROD servers, the path can remain the same and the contents of the files can differ depending on the environment.

Alternatively, without setting `twigkit.conf`, the configuration files from the WAR are used instead (from `WEB-INF/classes/conf`). You can use a feature like Maven build profiles to customize the WAR files that are built, however this requires more work and is only recommended if you are used to working in this manner or if you require a high degree of customization for each environment.

Localization Keys

This table identifies key names for UI labels and gives the default English labels.

For information about enabling translations, see *Enabling translations*.

Key	English Label	Additional Context
components.search-box.placeholder	Search…	Search input placeholder
components.search-box.button-label	Search	Search submit button label
components.search-box.saved	My Saved Searches	
components.search-box.bookmark	My bookmarks	
components.spelling-suggestions.did-you-mean	Did you mean: {correction}.	As in, "Did you mean *spelling-corrected query*?"
components.spelling-suggestions.auto-correct	Showing results for {correction}.	This is displayed above a list of results of a spelling-corrected query.
components.spelling-suggestions.no-results	No results found for {query}.	
components.response-statistics.showing	Showing {first} – {last} of {total}	
components.pagination.next	Next	
components.pagination.previous	Previous	
components.breadcrumbs.clear-all	Clear All	
components.facet.show-more	Show more	
components.facet.show-less	Show less	
components.no-results.title	Sorry: there are no results that match your search criteria.	
components.no-results.subtitle	If you have checked your spelling: try using more general keywords.	
components.response-statistics.showing-all	Showing All {total}	

Key	English Label	Additional Context
collaborate.bookmark.tooltip	Bookmark this?	As in, "Would you like to bookmark this item?"
collaborate.bookmark.active	You have bookmarked this	As in, this is already bookmarked.
collaborate.bookmark.single	{count} person has bookmarked this	As in the singular: "1 person has bookmarked this."
collaborate.bookmark.plural	{count} people have bookmarked this	As in the plural: "4 people have bookmarked this."
collaborate.bookmarkList.empty-text	You do not have any bookmarks yet	
collaborate.comment.prompt	Please enter a comment	As in a prompt to add a comment.
collaborate.comment.post	Post	The comment submit button
collaborate.comment.placeholder	Leave a new note…	The text area placeholder
collaborate.comment.delete	Delete	The delete comment button
collaborate.comment.more	Show more comments	
collaborate.comment.less	Show fewer	As in, show fewer comments.
collaborate.comment.delete-message	Are you sure you want to remove this comment?	
collaborate.like.tooltip	Do you like this?	As in, "Do you like this item/result?"
collaborate.like.active	You have liked this	
collaborate.like.singular	{count} person has liked this	As in the singular: "1 person has liked this."
collaborate.like.plural	{count} people have liked this	As in the plural "2 people have liked this."
collaborate.saved.button	Save	The save query button label

Key	English Label	Additional Context
collaborate.saved.empty-text	You do not have any saved queries yet	
collaborate.saved.error	Please enter a name so this query can be saved	Error message indicating a name is required
collaborate.saved.error-duplicate	A saved query with that name already exists, please use a different name	Error message flagging duplicate name
collaborate.saved.error-network	Unable to save query, please try again	General error message
collaborate.saved.placeholder	Search name	Guidance phrase in search box about searching names of saved searches
collaborate.topic.button	Save	The save topic button label
collaborate.topic.prompt	Please enter a topic name	
collaborate.topic.name	Type a name for the new topic…	The text input placeholder
collaborate.topic.description	Description…	Input placeholder for the topic description text
collaborate.topic.empty	You do not have any topics yet	
components.breadcrumbs.exclude	not {breadcrumb}	Showing the results that do not include items of this breadcrumb type
components.facet.apply-filters	Apply Filters	
components.facet.no-results	No results found	
components.facet.search	Search…	
content.form.success	Form posted successfully	
content.form.error	Error!{status}-{statusText}	
content.form.required	This field is required	

Key	English Label	Additional Context
`content.form.submit`	Submit	
`content.form.required-textarea`	This field is required	

Overlaying Configuration Values

To fine-tune your configuration hierarchy prior to production, Appkit has added the ability to overlay configuration hierarchies via the Jetty Maven plugin. While developing an application, you can experiment with a partial configuration and add/update only those properties that you actually must update without touching the application's main configuration. When the application starts these changes are merged with the application's main configuration hierarchy.

Example

To illustrate this, consider this snippet of (default) configuration for an application that in this case is using the Google Search Appliance (GSA):

```
/src /main /resources /conf /platforms /gsa gsa.conf platforms.conf /services
/suggestions static.conf services.conf
```

Typically, this configuration hierarchy will contain settings relating to a production environment. However, while the application is being developed, you might want to use different values for some of the configurations or introduce new values to test whether they can be moved over to the application's configuration hierarchy.

To facilitate this workflow, you can now configure your own hierarchy, which by default, can be placed in `src/dev/resources/conf`. Now, say, you want to use a different GSA host while developing your application but did not want to change the application's default configuration. To do this you could create this configuration hierarchy:

```
/src /dev /resources /conf /platforms /gsa gsa.conf
```

As stated, you can use the default location for your own configuration hierarchy. However, this location can be overridden at the command line, if needed, when launching the application, for example:

```
mvn clean jetty:run -Dtwigkit.conf.overlay="file://src/dev/my_resources/conf"
```

> You do not must fully populate your configuration hierarchy because this partial configuration will be merged with the application's configuration on application startup.

If the path to the user's configuration hierarchy cannot be found (for example, there is a mismatch between what is stated at the command line and what is actually in the project) then the application's default configuration hierarchy will be used.

Troubleshooting

If you are having issues loading your configuration hierarchy, try changing the logging level (in `src/main/resources/logback.xml`) to TRACE. Then look for entries in the log relating to the configuration hierarchy. Entries include `Secondary fig .. not found. Falling back to primary.` and likewise `Primary fig .. not found. Falling back to secondary`. The figs here refer to the application (primary) configuration hierarchy and your own (secondary) configuration hierarchy.

Structure of a Web Application

Appkit web applications follow a standard Java web project structure. If using Maven for build and dependency management, the application structure is as follows:

```
project-name/ `pom.xml` src/ main/ java/ resources/ conf/ webapp/ assets/ styles/ WEB-INF/ pages/ tags/
```

If deploying directly in a servlet container such as Tomcat rather than using Maven, however, the application structure is simply:

```
project-name/ assets/ styles/ WEB-INF/ classes/ conf/ pages/ tags/
```

Below, we take a closer look at four areas of the application: `pom.xml`, the assets and styles folders, the WEB-INF folder, and the resources or classes folder.

pom.xml

The build process is managed by the `pom.xml` file. It contains references to the Maven repositories and version information for the dependencies. When the application is built (using Maven) all dependencies are automatically retrieved, and a web application archive (WAR file) is generated. Configuration files are automatically placed in the correct folders within the WAR.

Assets and styles folders

The assets folder is where assets such as images are kept. The main logos in a standard Appkit application are stored and served from here.

The styles folder contains CSS and Less files that are specific to the application. The central stylesheet for configuring styling is classic-settings.less, which in turn includes several partial stylesheets that are stored in the `theme/partials/` folder.

Custom styling using Less can be added to `classic-settings.less` and the partial stylesheets, and custom CSS to the `custom.css` file.

WEB-INF folder

The WEB-INF folder includes pages, tags, and a handful of configuration files. It also includes the standard Java web application `web.xml` configuration file.

Pages folder

The pages folder is where the JSP files are kept. search.jsp or content.jsp is typically the main search page. Additional pages can be created here as needed.

Tags folder

The tags folder stores .tag files – commonly referred to as "widgets" – that are specific to the particular application. They can then be referenced in the JSPs using the client: prefix. Some of the default widgets include:

- **header.tag**. this widget contains the search header including the logo and search box

You can modify the existing .tag files as well as add your own.

Configuration files

In addition to the Appkit specific *configuration files* under 'conf' there are also these web application configuration files:

- **url-rules.xml**. This file is where URL routing can be configured. There is a default rule in place that says the /${foo}/ URL path should use the WEB-INF/pages/${foo}.jsp JSP page, hence /search/ would be routed to the page search.jsp.
- **wro.properties**. The application uses wro4j to optimize CSS and JavaScript resources for delivery. wro.properties can be used to specify which resources should be bundled together. Wro packages all the resources when the application starts, as well as recompiling on the fly if a CSS or JavaScript file is modified.

Resources or classes folder

The resources folder (if running with Maven/Jetty), or WEB-INF/classes if deploying with Tomcat, is the primary location for configuration files in the application. The `spring-security.xml` file contains security configuration. The `platform-cache.xml` file defines how responses from the search platform should be cached, and log4j2.xml controls the logging level.

Finally, there is the `conf` folder that provides a cascading property framework. Properties can be set at a general level (such as 'platforms.gsa') and then overwritten more specifically (for example, 'platforms.gsa.people'). These property trees are reported in the log at startup and can then be referenced throughout the application.

Navigation

The articles in this section describe Appkit tags used for navigation.

- *Breadcrumbs*
  The Breadcrumbs tag outputs the components of the Query for example, the query string and filters. If the original Query is provided to the tag via an attribute then HTML will automatically be generated to display and allow the removal of individual filters.
- *Faceted Navigation*
  Faceted navigation is one of the most powerful features of search. Appkit provides simple tags for rendering lists of facets and filters in your application, while also providing a single widget that handles many of the complexities of faceted navigation for you.
- *Tabs*
  The Tabs tag displays user interface components for tab-based navigation or filtering. The tag can be used to either output path-based tabs or filter tabs.

Breadcrumbs tag

The Breadcrumbs tag outputs the components of the Query for example, the query string and filters. If the original Query is provided to the tag via an attribute then HTML will automatically be generated to display and allow the removal of individual filters.

Usage

This example shows how to add breadcrumbs to the page. When no body is specified for the tag, Appkit will generate appropriate HTML.

```
<search:breadcrumbs response="response"></search:breadcrumbs>
```

For a complete list of attributes, see the *search:breadcrumbs* tag doc.

Faceted Navigation

Faceted navigation is one of the most powerful features of search. Appkit provides simple tags for rendering lists of facets and filters in your application, while also providing a single widget that handles many of the complexities of faceted navigation for you.

Listing facets

Self-closing the Facet List tag will output a list of all the facets and filter links with the default look-and-feel. *Find out more about the facetList tag*.

```
<search:facet-list response="response" facet-names="*"></search:facet-list>
```

You could then open the Facet List tag and place an individual Facet tag within it. *Find out more about the facet tag*.

```
<search:facet-list response="response" facet-names="*"> <search:facet facet-name="Author" collapsible="true"></search:facet> </search:facet-list>
```

This tag is also customisable to allow different display values and behavior. For a complete list of attributes, see the `search:facet-list` tag doc.

Tabs tag

The Tabs tag displays user interface components for tab-based navigation or filtering. The tag can be used to either output path-based tabs or filter tabs.

With path-based tabs, the tag outputs a fixed set of tabs that link to different URLs in the search application. This can be used to direct the user's search query between different views or collections, for example.

When displaying filter tabs, specific filters from a given facet are output even when the filter has no hits. This is often used for tab-based filtering, such as when the chosen facet partitions the search collections into different types of content. Be aware that when displaying filter tabs, this tag performs a second query to the search platform.

> The Tabs tag subsumes the functionality of the older Filter Tabs tag.

Usage for path-based tabs

This example shows how to display a fixed set of tabs, each linking to a different search URL. In this case, the tag outputs three tabs, labelled 'Content', 'People' and 'All content', which link to `<contextPath>/content`, `<contextPath>/people` and `<contextPath>`, respectively. When clicking on each of the three tabs, the current search query (the `query` object) is taken into account and appended to the URL that is generated.

```
<search:tabs query="query" tabs="content=Content,people=People" all="true" all-
label="All content"></search:tabs>
```

Usage for tab-based filtering

This example shows how to display filter tabs for a particular field. Filter based tabs are enabled by the addition of a `field-name="mediaType"` attribute, mediaType being the chosen field to filter on.

In this case both 'Videos' and 'Podcasts' filter will be grouped under the 'Multimedia' tab (using an OR query).

```
<search:tabs field-name="mediaType"
tabs="Articles,Blogs,Videos|Podcasts=Multimedia" all-label="Everything"
platform="platform" response="response" query="query" count-number="rounded">
</search:tabs>
```

For a complete list of attributes, see the *`search:tabs`* tag doc.

Rendering Search Results

Search results are the focus of every search interface, and Appkit makes it easy to forge the results precisely to your liking.

Result Lists

To add a list of results containing every available field, use a Result List tag on its own:

```
<search:result-list response="response"></search:result-list>
```

To choose which fields are displayed, add the Result tag inside the body of the Result List tag:

```
<search:result-list response="response"> <search:result> ... </search:result>
</search:result-list>
```

Working with fields

Inside the Result tag, use the Field tag to specify and configure each field:

```
<search:result-list response="response"> <search:result> <search:field name="title"
/> <search:field name="description" /> </search:result> </search:result-list>
```

Results Table

The Table tag library is a collection of JSP tags for rendering search results in table format.

To display a list of search results in a table, add the *table:result-list* tag on your search page, passing in a reference to the twigkit.model.Response object that contains your search results:

```
<table:result-list response="response"> <table:header> <table:header-cell
name="full_name" label="Name" sort="true"></table:cell> <table:header-cell
name="address" label="Address" sort="true"></table:cell> <table:header-cell
name="dob" label="Date of birth" sort="true"></table:cell> </table:header>
<table:body> <table:row> <table:cell name="full_name" query="query"></table:cell>
<table:cell name="address"></table:cell> <table:cell name="dob" date-
format="dd/MM/yyyy"></table:cell> </table:row> </table:body> </table:result-list>
```

Cell tag

The Cell tag displays the value of a given field from a search result as a table cell. The Cell tag supports a number of features in displaying the value, such as date formatting, adding a suffix or prefix, and default values.

Usage

Place the Cell tag inside the body of a *Row tag*. The order of the cell tags within a row should match the order of the *Header Cell tags* within the table's *Header tag*. In this example, a table contains three columns and displays the "full_name", "address", and "dob" fields for each document in the given search response.

```
<table:result-list response="response"> <table:header> <table:header-cell
name="full_name" label="Name" sort="true"></table:header-cell> <table:header-cell
name="address" label="Address" sort="true"></table:header-cell> <table:header-cell
name="dob" label="Date of birth" sort="true"></table:header-cell> </table:header>
<table:body> <table:row> <table:cell name="full_name" query="query"></table:cell>
<table:cell name="address"</table:cell> <table:cell name="dob" date-
format="dd/MM/yyyy"</table:cell> </table:row> </table:body> </table:result-list>
```

For a complete list of attributes, see the *table:cell* tag doc.

Header Cell tag

The Header Cell tag outputs a single header cell in a results table. The tag must be nested within a *Header tag*.

Usage

Place the Header Cell tag inside the body of a *Header tag*. The order of the cell tags within a header should match the order of the *Cell tags* within the table's *Row tag*. In this example, a table contains three columns and displays the "full_name", "address", and "dob" fields for each document in the given search response.

```
<table:result-list response="response"> <table:header> <table:header-cell
name="full_name" label="Name" sort="true"></table:header-cell> <table:header-cell
name="address" label="Address" sort="true"></table:header-cell> <table:header-cell
name="dob" label="Date of birth" sort="true"></table:header-cell> </table:header>
... </table:result-list>
```

For a complete list of attributes, see the *table:header-cell* tag doc.

Header tag

The Table Header tag outputs the header of the results table. The tag must be nested within a *Table Result List tag*.

Usage

Place the Header tag inside the body of a /app-studio/5.4/3103/table-result-list-tag[Table Result List tag]. It contains nested *Header Cell tags*, one for every table column.

```
<table:result-list response="response"> <table:header> <table:header-cell
name="full_name" label="Name" sort="true"></table:header-cell> <table:header-cell
name="address" label="Address" sort="true"></table:header-cell> <table:header-cell
name="dob" label="Date of birth" sort="true"></table:header-cell> </table:header>
... </table:result-list>
```

Row tag

The Table Row tag specifies how rows of the results table should be formatted. The tag must be nested within a *Table Body tag*.

Usage

Place the Row tag inside the body of a *Table Body tag*. It contains nested *Cell tags*, one for every table column.

```
<table:result-list response="response"> ... <table:body> <table:row> <table:cell
name="full_name" query="query"></table:cell> <table:cell name="address">
</table:cell> <table:cell name="dob" date-format="dd/MM/yyyy"></table:cell>
</table:row> </table:body> </table:result-list>
```

Table Body tag

The Table Body tag contains the main body of the results table. The tag must be nested within a *Result List tag*.

Usage

Place the Body tag inside the body of a *Result List tag*. It contains a *Row tag*, to specify the format of the table rows.

```
<table:result-list response="response"> ... <table:body> <table:row> ...
</table:row> </table:body> </table:result-list>
```

For a complete list of attributes, see the `table:body` tag doc.

Table Result List tag

The Table Result List tag displays the results for a given search response in a table. The body of the Table Result List tag contains a *Header* tag which outputs the top header of the table, and a *Body* tag, which specifies the format of the rows of the table.

Usage

This example shows how to display all results in a search response in a table.

```
<table:result-list response="response"> <table:header> ... </table:header>
<table:body> ... </table:body> </table:result-list>
```

For a complete list of attributes, see the *table:result-list* tag doc.

Result List tag

The Result List tag displays the results for a given Response. While it can be used as a standalone tag to output every field for each result, it is most often contains a Result tag with numerous Field tags.

Usage

This example shows how to display all results returned by the Platform. It shows two alternative methods of referencing the fields of each item:

- Using angular syntax to use the value directly
- Using the Field tag to produce strictly valid XHTML with comprehensive CSS selectors.

```
<search:result-list response="response"> <search:result> <h2><a ng-href="{{result | field:'url' | actual}}" target="_top">{{result | field:'title' | actual}}</a></h2> <search:field name="body" label=""></search:field> <search:field name="concepts" label="concepts" query="query" separator=";"></search:field> </search:result> </search:result-list>
```

While not strictly required, the nested `<search:result>` tag is recommended. For more details, refer to the *Result tag*.

Nested Result Lists

Some search platforms, such as Solr (when using Grouping) will create Results with further nested Results. In Appkit terminology these are named 'related results'. A JOIN will produce a secondary set of hits for each result in the response. This secondary set of hits can be accessed and output using a nested Result List tag:

```
<search:result-list response="response"> <search:result> <search:field name="country" label="" value-element="h2"></search:field> <search:field name="text" label=""></search:field> <search:field name="climate" label="Climate" query="query"></search:field> <h3>Cities in {{result | field:'title' | actual}} </h3> <!-- Nested list of related Results --> <search:result-list results="result.related"> <search:result> <search:field name="city" label=""> </search:field> </search:result> </search:result-list> </search:result> </search:result-list>
```

See the *search:result-list* tag doc for all available attributes.

Tag body variables

The body of the tag will be repeated for every result in the Response.

${result} (twigkit.model.Result)
The result being output. See the Result for available properties.

- To get a String representation of a Field use: `{{result | field:'name' | actual}}` or `{{result | field:'title' | display}}`.

Result tags

The Result tag renders if results are available. It outputs a single result, and must be nested within a Result List tag. The No Results tag renders if no results are available.

Result tag

The Result tag renders if results are available. It outputs a single result, and must be nested within a Result List tag.

```
<search:result-list response="response"> <search:result> <search:field name="title"></search:field> <search:field name="description"></search:field> </search:result> </search:result-list>
```

See the *search:result* tag doc for all available attributes.

Conditional result tag

It is also possible to use the result tag to conditionally display a set of fields based on the value of a given field. This is useful when each result fit into a specific *category*.

Usage

The syntax is as follows:

```
<search:result-list response="response"> <search:result a-field-name="fieldValue"> <search:field name="title"></search:field> <search:field name="description"> </search:field> </search:result> <search:result a-field-name="otherFieldValue"> <search:field name="otherTitle"></search:field> <search:field name="otherDescription"></search:field> </search:result> </search:result-list>
```

For example, if you had 2 types of results, documents and people, with different fields, you could display them in a single result list as follows:

```
<search:result-list response="response"> <search:result type="people"> <search:field name="name"></search:field> <search:field name="date-of-birth"> </search:field> </search:result> <search:result type="document"> <search:field name="title"></search:field> <search:field name="author"></search:field> </search:result> </search:result-list>
```

It is also possible to combine multiple conditions. All have to be met for the specific <search:result> tag to be selected. For example:

```
<search:result-list response="response"> <search:result type="people"
country="Vietnam"> <h1>Vietnam</h1> <search:field name="name"></search:field>
<search:field name="date-of-birth"></search:field> </search:result> <search:result
type="people"> <h1>Rest of the World</h1> <search:field name="name"></search:field>
<search:field name="date-of-birth"></search:field> </search:result>
</search:result-list>
```

Extending Appkit

The Appkit application framework adheres to standard Java web application methodologies making custom development using generally accepted principles perfectly acceptable. In addition to that Appkit provides a number integration hooks for any aspect of the lifecycle and extension points for adding custom capabilities and features to Appkit application.

Fundamentals

The Appkit software stack is a 100% Java framework and most commonly deployed as a standard Java Web Application using best practice (and non-prescriptive) conventions. The framework is stateless with (in the case of web applications) all parameters of a Query generated from HTTP request parameters.

Appkit uses Google Guice for dependency injection and plugin architectures and Maven for dependency and build management. The Appkit software components are available via our private Maven repository: `http://repo.twigkit.com/`. Use the settings.xml *(https://dev.twigkit.net/downloads/settings.xml)* file to correctly resolve dependencies to the Appkit libraries.

Appkit SDK

The Appkit SDK is a fully configured, deployable software project with placeholders for the major components described above. For further information submit questions via the issue tracking system.

Collaboration Events

All social and collaboration events relating to annotations (likes, comments etc.) are published to a local event bus. This means that these events can be subscribed to and custom behavior can be implemented when the event is fired. For example, the custom behavior might be invoking some code that submits information to additional systems or that triggers a message such as an email notification.

To do this, create a listener for these events. Here is an example to do this for 'likes'. Add this code to `src/main/java/twigkit/events/AnnotationInterceptorModule.java`:

```
package twigkit.events; import com.google.common.eventbus.Subscribe; import org.slf4j.Logger; import org.slf4j.LoggerFactory; import twigkit.AbstractTwigKitModule; import twigkit.event.InjectorReady; public class AnnotationInterceptorModule extends AbstractTwigKitModule { private static final Logger logger = LoggerFactory.getLogger(AnnotationInterceptorModule.class); public AnnotationInterceptorModule() { super(Priority.LOWEST); } @Override protected void configure() { eventBus.register(new Object() { @Subscribe public void injectorReady(InjectorReady event) { try { eventBus.register(new AnnotationInterceptor()); } catch (Exception e) { } } }); logger.info("Annotation Interceptor application module loaded."); } }
```

And also create this file in `src/main/java/twigkit/events/AnnotationInterceptor.java`:

```
package twigkit.events; import com.google.common.eventbus.Subscribe; import org.slf4j.Logger; import org.slf4j.LoggerFactory; import twigkit.TwigKitApplication; import twigkit.social.event.AnnotationCreated; import twigkit.social.model.Annotation; import twigkit.social.model.Link; public class AnnotationInterceptor { private static final Logger logger = LoggerFactory.getLogger(AnnotationInterceptor.class); public AnnotationInterceptor(){ } /** * * @param event social event picked from the event bus */ @Subscribe public void annotate(AnnotationCreated event) { Annotation annotation = event.getAnnotation(); if (annotation instanceof Link) { logger.debug("This is a like annotation); //DO SOMETHING } } }
```

You must then register your new module by adding `twigkit.events.AnnotationInterceptorModule` to the file `src/main/resources/META-INF/services/twigkit.TwigKitModule`. If this file does not already exist simply create it and add the text `twigkit.events.AnnotationInterceptorModule`.

This is all that is required to implement custom behavior. Add the code you must the annotate method using the information provided in the `AnnotationCreated` event object. As with other custom modules registered in Appkit you can also inject references to platforms and other core Appkit components and interact with them here.

Developing Platforms

Platforms wrap the underlying search platform; creating a unified abstraction layer for major platform operations such as search, load, delete and store. You can add your own Platform implementation that wrap arbitrary data providers with a few simple lines of code and configurations.

Process Flow

At a high level the process flow for search operations is simple: A Query is submitted to a Platform, which it turns into a request to the underlying data repository. The Platform implementation then turns that into an Appkit Response and returns.

```
public class SomePlatform implements Platform { public Response search(Query query) { ... } }
```

Configurability

Typically the platform will require configuration and initialisation parameters, which can be passed on for example, using the Generic Platform tag. A simple way of achieving this with your platform is to extend the AbstractPlatform class:

```
package acme; public class SomePlatform extends AbstractPlatform { public Response search(Query query) { if (hasParameter("foo")) { String foo = getParameterStringValue("foo"); ... } } }
```

This means that the platform can be instantiated using the configuration framework or in the user interface JSP page with:

```
<search:platform className="acme.SomePlatform" foo="bar" />
```

For a more complex or flexible implementation, the process of adapting source queries and responses is typically encapsulated in distinct components, using an Adapter Pattern.

In this case a Platform implementation is injected with a Query Adapter and a Response Adapter which will handle the mapping of Appkit queries to underlying search platform queries (for example, a SolrQuery) and source responses (for example, a Solr QueryResponse) to an Appkit Response.

```
public class SomePlatform extends AbstractPlatform { @Inject public SomePlatform(@SomePlatform QueryAdapter<SomeQuery> queryAdapter, @SomePlatform ResponseAdapter<SomeResponse> responseAdapter) { ... } public Response search(Query query) { Response response = new Response(); SomeQuery sq = queryAdapter.map(this, query); ... Response response = responseAdapter.map(this, query, response, someResponse); return validate(query, response); } }
```

The AbstractPlatform also provides a comprehensive set of validation procedures for dealing with spelling suggestions, auto-corrections, and other aspects of the

query/response lifecycle. Furthermore, the `twigkit.platform.AbstractQueryAdapter`and`twigkit.platform.AbstractResponseAdapter` classes help you to ensure that you map all of the critical components, so that the Appkit widgets can effectively work with the new data source.

After you have done this, your platform is a fully fledged member of the framework which will act and behave like any other supported platform, meaning that it will support the full range of features and can be plugged into any Appkit application, federated to, etc.

Deploying your Platform

You can refer to your platform by fully qualified class name in the Platform tag, as long as it is present on the application's classpath.

A more efficient method would be to package it so that it will be automatically instantiated and provided by Guice. To do that use for example, Maven to create a jar file for the platform, making sure this folder and file structure is included:

```
/META-INF/ /services/ twigkit.platform.Platform <-- this is a file
```

The `twigkit.platform.Platform` file should contain a line delimited list of fully qualified class names of all the platforms in your jar. Doing this means that on start-up Appkit will dynamically load all available platforms and make them available for use via Guice injection.

Developing Processors

The Appkit Workflow module can be used to apply custom business logic during the search/response/render lifecycle. Workflow Processors are designed to intercept any Appkit model object, and for example, modify a Query before it is submitted to a Platform or augment a Response before it is rendered.

Developing Processors

All processors implement the Processor interface, parameterized with the object they are designed to work with:

```
public class AcmeProcessor implements Processor<Query> { public void init() { ... }
public void process(Query query) { ... } public void done() { ... } }
```

Or:

```
public class AcmeProcessor implements Processor<Response> { public void
process(Response response) { ... } }
```

Processors also support the change(T obj) method which will let you substitute model objects: `public Response change(Response response);`

> If you extend an AbstractProcessor class then change() is automatically named by the process() method making it sufficient to implement the former.

A number of abstract helper classes make it easier to implement processors for a given purpose, such as:

`twigkit.processor.AbstractProcessor<T>`
Provides abstract implmentations of init() and done() and handles setting parameters.

`twigkit.search.processors.AbstractResultProcessor`
Makes it easy to implement processors for Results by taking care of the iterations
`public void processResult(int index, Result result) {…}`

`twigkit.search.processors.AbstractNamedFieldProcessor`
Makes it easy to implement processors that target specific fields. Requires the `fields` parameter (use * for all fields). Only the named fields will be passed on to the process method.
`public void process(Field field) {…}`

`twigkit.search.processors.AbstractNamedFieldValueProcessor<T>`
Use this abstract class to implement a Processor that will pass the named field values (where the value is an instance of T). Requires the fields parameter (use * for all fields). Validation of the field value being of the correct type is done by the abstract class.
`public Value<String> processValue(Value<String> value) {…}`

`twigkit.search.processors.AbstractNamedFacetProcessor`
Use this abstract class to implement Processor that will process specific named facets. Requires the facetNames parameter (use * for all facets).
`public void processFacet(Facet facet) {…}`

`twigkit.search.processors.AbstractNamedFacetFiltersProcessor`
Use this abstract class to process all the FacetFilters in the named facets. Requires the facetNames parameter (use * for all facets).
`public void processFilter(FacetFilter filter) {…}`

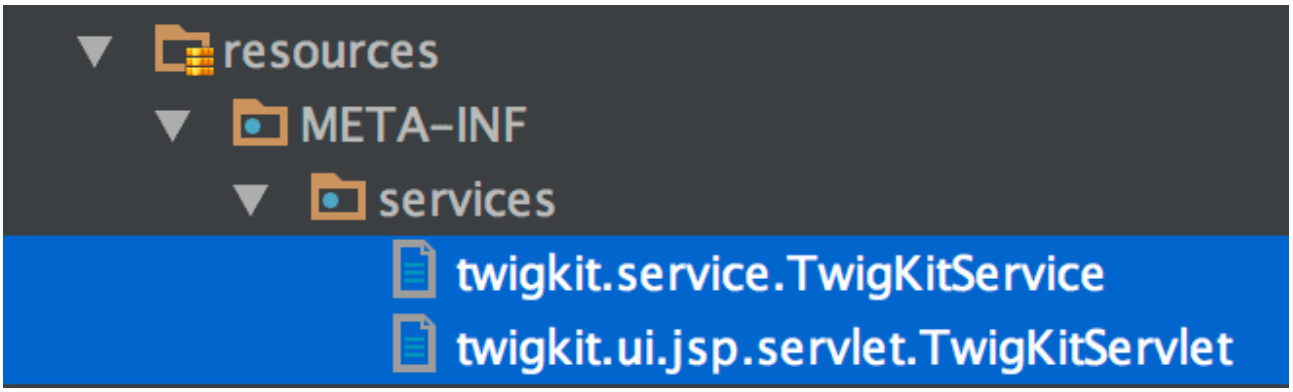
Developing Services and Servlets

You can develop services and servlets:

- Services encapsulate reusable processes that can be injected into providers such as Servlets. For example, different implementations of Query Suggestions are implement the `twigkit.service.AppkitService` interface and can be injected into the servlet responsible for returning the list of suggestions.
- Besides using standard JSP to generate output; Servlets are common way of doing custom HTTP responses with Java. You can deploy standard Java Servlets in Appkit application but in addition to that Appkit provides a convenient framework for developing and deploying servlets for search based applications.

Services

Services can be instantiated programmatically or using the Appkit configuration framework, making them available to other processes or programmatically via the `ConfiguredServiceProvider` class. To declare a service via the configuration framework, you must add a configuration file named `src/main/resources/META-INF/services/TwigKitService` that contains the fully-qualified Java class name of your services, one per line. See the screenshot for an example.



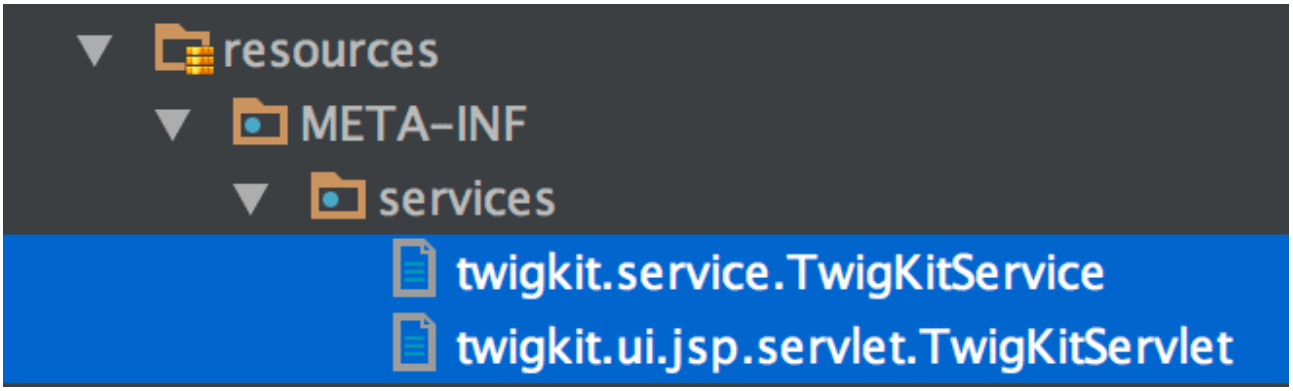
> For legacy reasons, the service file must be named `TwigKitService`, with a capital "K".

Below is a sample signature for a Query Suggestion Service:

```
public class StaticQuerySuggestionService extends Parameters implements QuerySuggestionService, TwigKitService { public List<QuerySuggestion> suggest(final Query query) { // Do whatever here to generate a List of QuerySuggestions for the Query supplied! } }
```

Servlets and custom output formats

An instance of an AppkitServlet is automatically discovered and deployed by the Appkit application on startup, as long as your class is annotated with the path you would like it published at, implements or extends a `AppkitServlet`, and is declared via the Appkit configuration framework. To declare a servlet via the configuration framework, you must add a configuration file named `src/main/resources/META-INF/services/TwigKitServlet` that contains the fully-qualified Java class name of your servlets, one per line. See the screenshot for an example.



For legacy reasons, the service file must be named `TwigKitServlet`, with a capital "K". Below is an example of a `TwigKitServlet`.

```
@ServletPath(path = "/custom/excel/{platform}") public class SomeServlet extends TwigKitServlet { protected void doGet(HttpServletRequest req, HttpServletResponse resp, Query query, Config conf) throws ServletException, IOException { // Query instances and Appkit configurations are automatically provided to your method! } }
```

This means that the process of changing stateless HTTP requests to queries is handled for you, and the Servlet is automatically declared and published, in this case to respond at this URL: `/twigkit/services/custom/excel/`. Using the ServletPath annotation will guarantee that the HTTP Request/Response is correctly filtered by Appkit for example, for security and URL mapping.

Layout and Styling

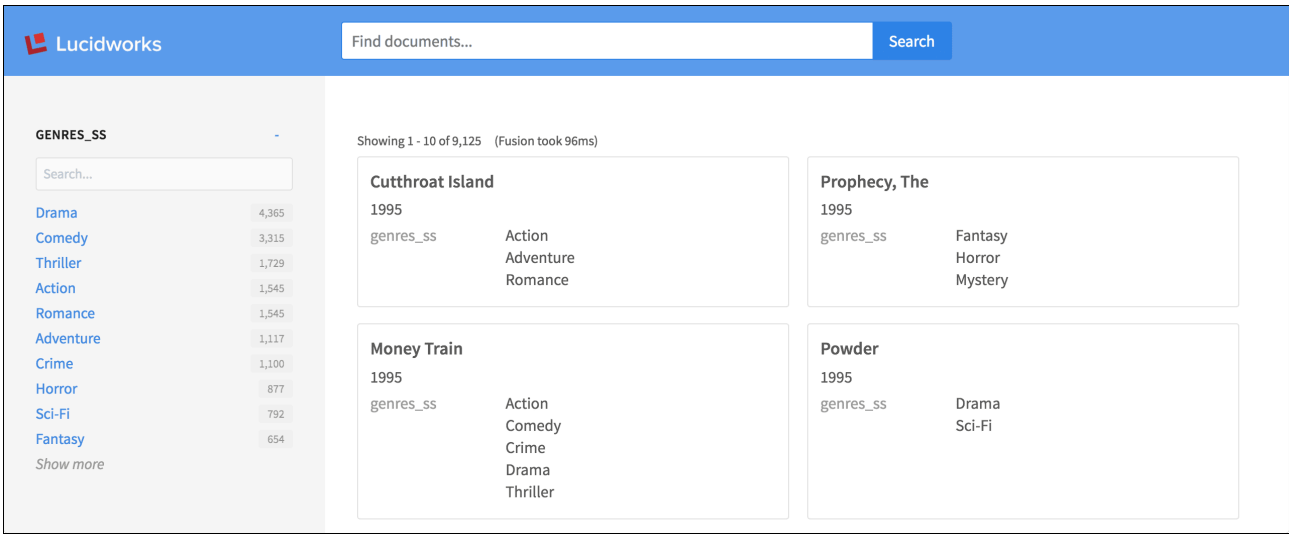
The articles in this section describe the basics of layouts and styling.

- *Layout of the Default View*
  This topic explores the elements that compose the default App Studio view.
- *Themes and Styling*
  Appkit is designed so that themes can be used to apply a different look and feel to an application. These encompass all visual aspects of a search user interface and currently there is one main *classic* theme which every project starts out with.
- *Topic Pages*
  Appkit has advanced support for specifying custom URL patterns for filtering and grouping search results.

Layout of the Default View

This topic explores the elements that compose the default App Studio view.

The default view looks like this:

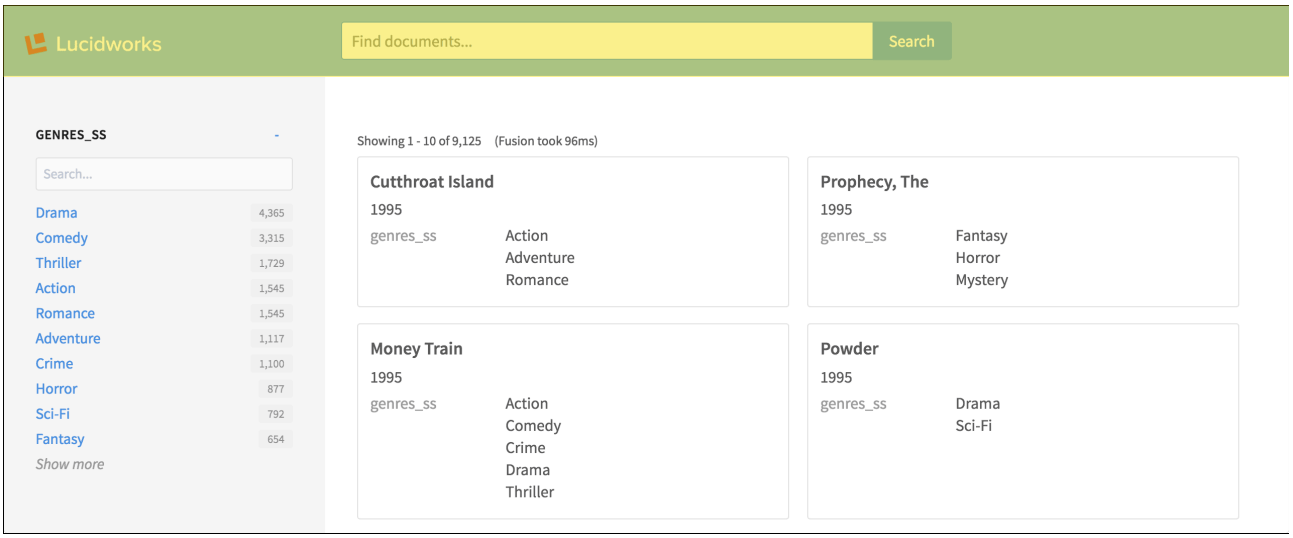


As we explore the components of the page, we will highlight each area in turn.

Include a partial file

The header of the default view is a partial file that is included with the `layout:include` tag:

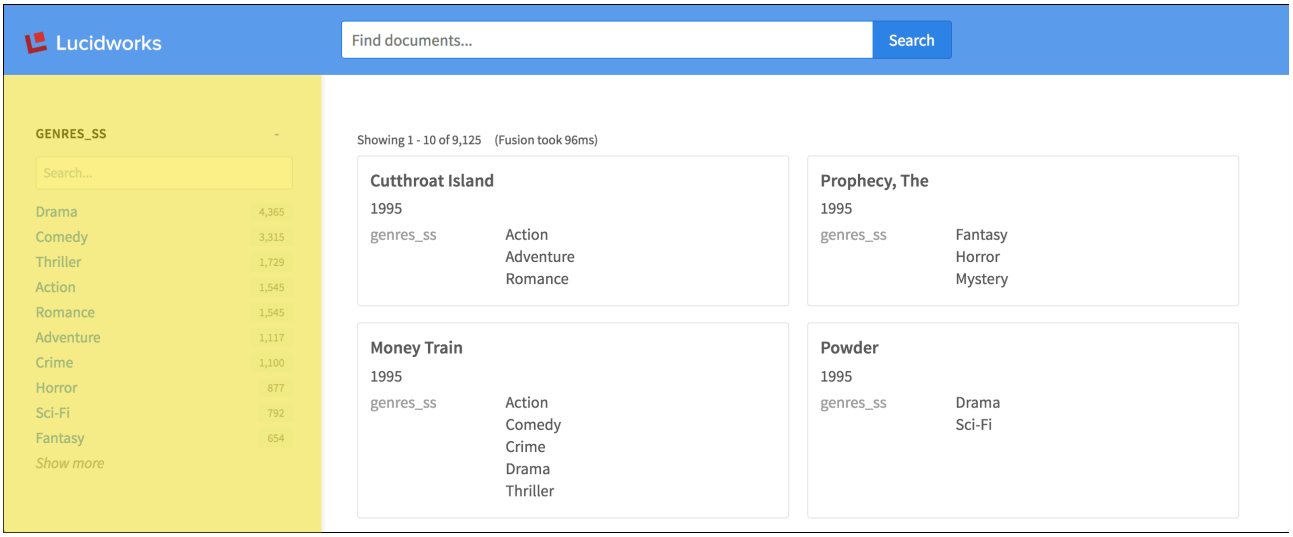
```
<layout:include file="views/partials/header.tpl.html" action="search">
</layout:include>
```



The `header.tpl.html` file, in turn, contains a number of `layout` tags, plus the important `search:box` tag.

Sidebar with facet list

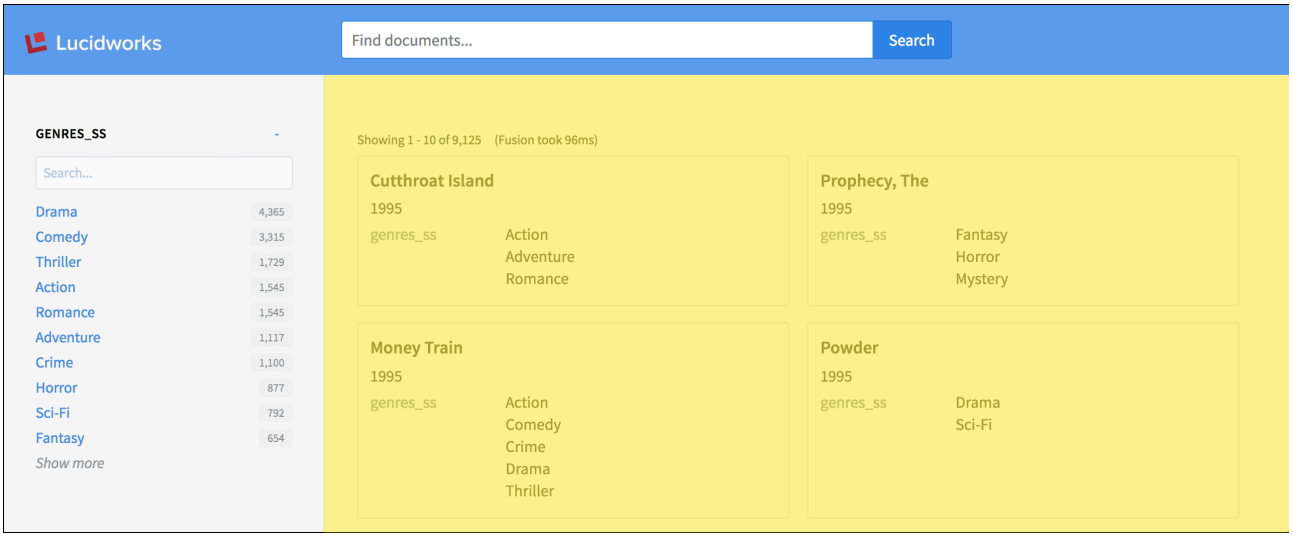
The sidebar is rendered using the simple `layout:sidebar` tag. Within the sidebar, the `search:facet-list` and `search:facet` tags contain the parameters that configure the facet fields and their display.




```
<layout:block md="1-3" lg="1-4" drawer="left" id="sidebar" styling="blocksidebar-light"> <layout:sidebar> <layout:box> <!-- List facets and give them intuitive labels --> <search:facet-list facet-names="genres_ss" response="response" platform="platform" query="query" styling="facet-list facet-list-wrappedheader"> <search:facet when-field="genres_ss" show="10" search-enabled="true" platform="platform" query="query" max-characters="40" show="12" show-more="24" collapsible="true"></search:facet> <search:facet show="10" max-characters="40" show="12" show-more="24" collapsible="true"></search:facet> </search:facet-list> </layout:box> </layout:sidebar> </layout:block>
```

Search statistics and results list

In the content area, `layout` tags are used again to construct the grid layout. For brevity, we will omit those from the code samples below.



The `search:response-statistics` tag displays the number of search results, and a `{{ response.time }}` variable displays the number of milliseconds Fusion took to respond:

```
<search:response-statistics response="response"></search:response-statistics> <p class="response-statistics">(Fusion took {{ response.time }}ms)</p>
```

Then, the *search:result-list* tag lays out the grid of search results:

```
<search:result-list response="response" styling="cards-sm-1 cards-lg-2 cards-xl-2" platform="platform" query="query" instant-results="true"> ... </search:result-list>
```

Individual results

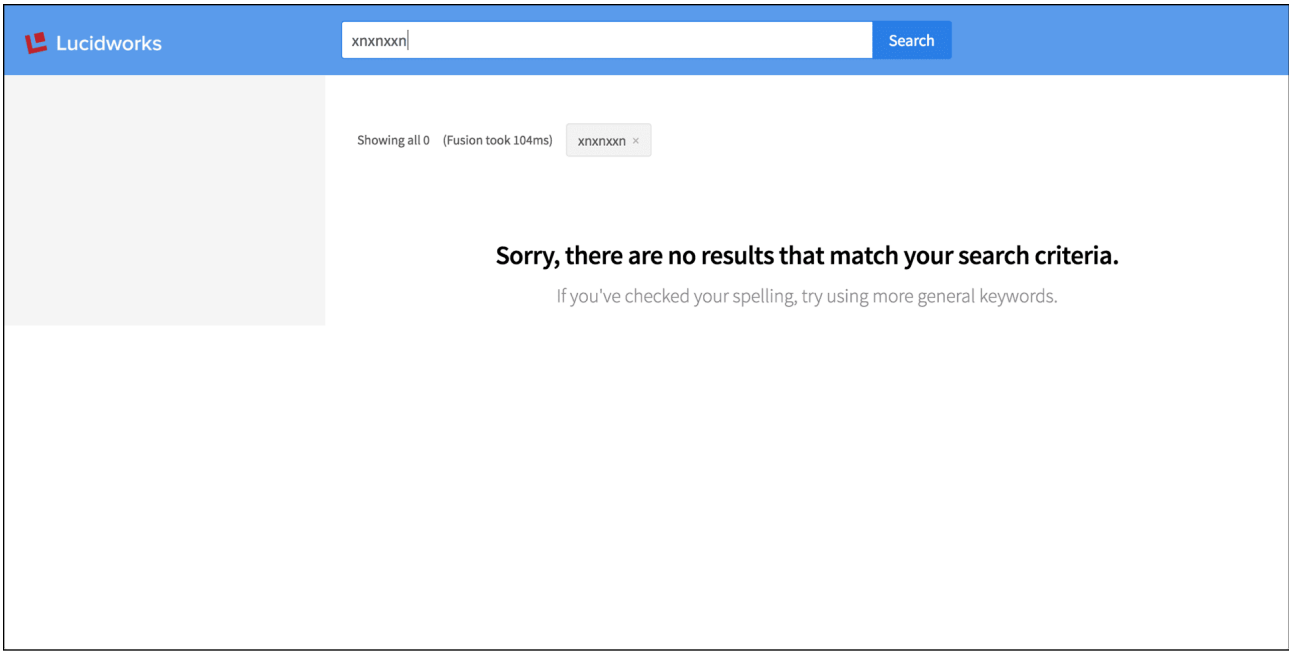
The *search:result* and *search:field* tags configure which fields are displayed with each individual search result. A field can be styled as the title or description, and any additional fields can have a variety of styles.

```
<search:result> <search:field name="title_txt" styling="title" urlfield> </search:field> <search:field name="year_i" styling="description" max-characters="150"></search:field> <search:field name="genres_ss" styling="label-left"></search:field> </search:result>
```

Null search results

In the case of no search results, the search:no-results tag displays a default message:

```
<search:no-results response="response"></search:no-results>
```


Lucidworks
xnxnxxn
Search
Showing all 0 (Fusion took 104ms)
xnxnxxn ×
Sorry, there are no results that match your search criteria.
If you've checked your spelling, try using more general keywords.

Themes and Styling

Appkit is designed so that themes can be used to apply a different look and feel to an application. These encompass all visual aspects of a search user interface and currently there is one main *classic* theme which every project starts out with.

Themes can be controlled via the dependencies specified in the maven `pom.xml` file and will be available to select when a new project is generated through Launchpad.

Styling

Each of the themes uses a main Less file in `src/main/webapp/styles` named `themeName-settings.less`. This provides a basis for modifying the parameters of a theme at a high level such as the color scheme. If you are not familiar with Less or want to customize the UI by applying some standard CSS styling, this can be done in the `custom.css` file in the same location.

When to use Less and when to use CSS

In some cases, it is easier to customize particular components of the UI through the Less settings, rather than `custom.css`. In our themes and styling, Appkit defines a number of Less “mixins” which are re-usable code snippets that can be invoked using some number of parameters. The Less code is never exposed to the web client directly, but rather is compiled into CSS code at runtime (or even at compile time). When compiled, a single line of a `.less` file can result in multiple lines of CSS.

This means that in order to customize the default styling of a theme, you can:

- Modify a few lines in the `classic-settings.less` file
- Override the compiled CSS settings in the custom.css file.

This is best explained with an example. Consider a Less file with this “makeRed” re-usable code mixin:

```
.makeRed () { color: red; }
```

You can apply this mixin later by invoking it through a CSS rule; for example:

```
p { makeRed(); border: 1px black; }
```

which would output this in the compiled CSS file

```
p { color: red; border: 1px black; }
```

Looking at a particular Appkit example, consider for example, the way in which we specify the “title” field styling in the `classic-settings.less` file:

```
.tk-stl-title { .field-title(); } // @field-value-color, @field-label-color
```

Here, we use a `.field-title` mixin, which is defined in the classic theme. The comment at the end of the line lists the arguments that can be specified in this particular mixin. When compiled, the output CSS includes these rules:

```
.tk-stl-title .label { color: #000000; } .tk-stl-title .value { font-weight: bold; margin: 0 0 2px 0; font-size: 1.23em; line-height: 1.1667em; color: #52cc95; } .tk-stl-title .value a { color: #52cc95; }
```

To override the field title styling (`.tk-stl-title`), you could either change that one line in the Less file or include rules in `custom.css` overriding one or more of the compiled CSS rules, above.

So to conclude, it makes sense to override the Less settings when you are customising a part of the UI that has a well-defined mixin to match. For ad-hoc customization, it might be easiest to use `custom.css` directly.

Topic Pages

Appkit has advanced support for specifying custom URL patterns for filtering and grouping search results.

The Topic Pages module includes specialized versions of tags that work together to:

- Output links with a particular path structure according to a pattern
- Interpret the path of the current address to apply filters to a search.

This allows for a rich user interface which can help a user make sense of highly 'categorized' data and map different views of the data to readable URLs.

See also *Add a Detail Page*.

Appkit SDK Release Notes

Latest Release: 4.14.0 (31 October 2022)

Read the full release notes for version 4.14.0.

Improvements

- Swapped out dependencies on Log4j 1.2.x with SL4J in the Appkit code base.
- Fix for a bug that would prevent interval facets from toggling. Fix requires the `facet-list` tag to not use the `query=""` attribute.
- Fix for an issue when using the `<widget:spelling-suggestions>` component that would allow a cross-site scripting attack if the query term included a `<script>` tag and the Fusion `spellcheck` component returned a spell correction.

Previous releases

4.13.0 (15 September 2021)

Read the full release notes for version 4.13.0.

- Upgraded AngularJS framework to 1.8.2
  - Version 1.8.2 meteoric-mining corrects `$sceDelegate` to ensure that `resourceUrlWhitelist()` is identical to `trustedResourceUrlList()`.
- Upgraded JQuery to version 3.60.
  - Version 3.60 corrects security issues during a pen test.
- Fixed a bug where the selection of a query:suggestion component added the following to the URL: an `mq` parameter and an `mq value` surrounded by quotes.
- Fixed a bug where the facet filter for a facet already applied did not clear when `select=toggle`.
- Fixed a bug in the search:box with the action attribute where the queries on the search submission page were re-triggered *before* the page was redirected to the value listed in the action attribute.
- Fixed an issue where extra requests were made when a new query was submitted *if* a fixed header and infinite scroll were used.
- Fixed an issue to ensure a facet filter that contains special characters cannot be deselected.

4.11.0 (21 July 2020)

Read the full release notes for version 4.11.0.

- Ability to limit the results displayed by the result-list to a subset of the array
  - For example, the following: `<result-list response="response" first="2" last="5">` will display the 3rd, 4th and 5th results in the response.
- ES7 platform can now be configured whether or not to track total hits.
  - In the ES7 platform config (for example, resources/conf/platforms/elastic/elastic.conf) you can now add this config to track total hits: `track-total-hits: true`
  - By default, this will be **false** and will max out at **10000** as stated in the ES docs.
- It is now possible to specify whether to include or exclude the range filter boundaries by specifying the fromExcluded and toExcluded attributes to the `<query:filter>` tag.
  - For example:

```
<search:query parameters="*" var="query"> <query:filter field="age" from="25" to="50" display="25 to 50" fromExcluded="true" toExcluded="true">
</query:filter> </search:query>
```

- Appkit no longer adds the apollo path prefix to Fusion API URLs. If still required, the path prefix may be set via configuration.
- Fixed a bug where `<search:response-statistics>` are only updated when its specified response object is updated, not on any others.
- Added support for Angular expressions as a value for the Bookmark-list `collection` parameter.
- `HighlightingUtils` adds highlighting terms support when closing braces are absent.
- Fusion pre-authentication filter now uses the JWT token subject field for the user name.
- In `collaborate:saved-query-list`, the `show-more` user click option now specifies the number of additional list items to display.

4.10.0 (27 April 2020)

Read the full release notes for version 4.10.0.

- A new attribute, `auto-populate`, is added to the `<collaborate:saved-query-list>` tag. When set to `true`, the saved search name is automatically pre-populated with the query term in the URL. The default value for this attribute is `false`.
- Fixed a bug that sometimes caused selected filters to be removed from a query after applying new filters on the same facet.
- Fixed a bug that sometimes returned invalid annotation IDs when using the social response processors, which prevented the end-user from further deleting the annotation (like a bookmark or saved-search).
- Specifying how many annotations, such as bookmarks or saved-searches, to retrieve using the Social API would not result in the expected behavior. This has now been fixed.

4.9.0 (27 September 2019)

Read the full release notes for version 4.9.0.

- The `FusionProxy` endpoint `/twigkit/api/fusion/*` now accepts nested URI paths as expected. For example:

  ```
  /insights/twigkit/api/fusion/collections/<collection_name>
  /insights/twigkit/api/fusion/apps/<app_name>/experiments
  ```

  Previously, an `HTTP 400` error occurred.
- Removing filters using the `<query:filter action="remove">` tag will no longer remove filters added with the `<query:filter value="bar">` tag. For example, the following will leave the filter on `value="bar"` intact:

  ```
  <search:query var="query"> <query:filter field="foo" value="bar"> <query:filter
  field="foo" action="remove"> </search:query>
  ```

- Previously, using the Solr platform in compatibility mode forced exclusive boundaries on date range filters to become inclusive. This affected date range filters in the Fusion platform.
  This behavior is fixed. Exclusive boundaries on range filters now remain exclusive in compatibility mode.
- Toggle events now fire as expected when using toggle controls.

4.8.0 (10 January 2020)

Read the full release notes for version 4.8.0.

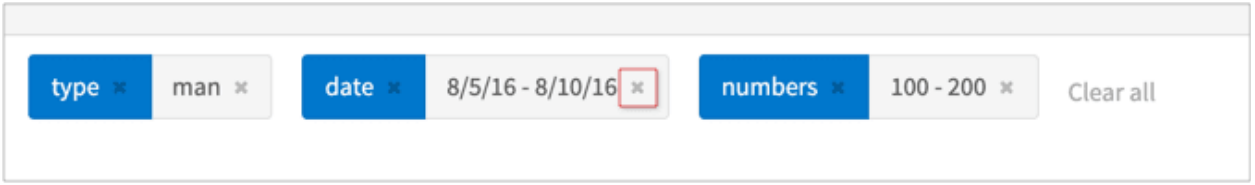
- It is now possible to map a user's LDAP roles to the application user's group. This differs from the previous implementation to the AD pre-authorization filter, in that it reads the groups directly from the user object instead of querying for AD groups that the user belongs to.
  To enable this filter, add a configuration file under `conf/security/ldap/provider.conf` with the following parameters:

```
enabled: true user: #Service account used to connect to LDAP. password:
#Password for the service account. This can be encrypted using the twigcrypt
binary. url: #Full URL to the LDAP server. user-search-base: #LDAP search base
to restricted the search to. user-search-filter: #LDAP search filter to restrict
the LDAP search.
```

- It is now possible to use basic authentication against a Solr cloud platform using the parameters `userName` and `password` in the platform configuration.
- The value of a highlighted field is now stored as the "display" field value, rather than replacing the original "actual" field value.
- Appkit 4.7.0 adds support for Elastic 7.4.
- A new filter, `find`, is added which searches the scope hierarchy to find a specified variable. For example: `{{ 'myVariable' | find:this }}`.
- The `twigkit.http.filter.DevelopmentErrorFilter` is replaced by the `twigkit.http.filter.ErrorFilter`. If you are using the `twigkit.http.filter.ErrorFilter` parameter in your `web.xml` file and an AJAX request fails, a JSON response is received identifying the status code of the REST call and any underlying causes. Previously, an HTML response was received.
  Example configuration:

```
<filter> <filter-name>ErrorFilter</filter-name> <filter-
class>twigkit.http.filter.ErrorFilter</filter-class> <init-param> <param-
name>responseCode</param-name> <param-value>400,405,406,500,501,503</param-
value> </init-param> <init-param> <param-name>excludedPaths</param-name> <param-
value>/,/login,/login/</param-value> </init-param> </filter>
```

- Accessibility improvements are made to the *search breadcrumbs*. The text, `x`, that removed the breadcrumb is replaced by an icon with the alt-text "Remove this term from the query" for use by screen readers.



Older releases

- *Appkit SDK 4.7.0.*
- *Appkit SDK 4.6.0*
- *Appkit SDK 4.5.0*
- *Appkit SDK 4.4.0*

- *Appkit SDK 4.3.0*
- *Appkit SDK 4.2.5*
- *Appkit SDK 4.2.4*
- *Appkit SDK 4.2.3*
- *Appkit SDK 4.2.2*
- *Appkit SDK 4.2.1*
- *Appkit SDK 4.2.0*

Appkit SDK 4.14.0 Release Notes

Release date: 31 October 2022

Additional upgrade steps

Upon upgrading, users should remove `src/main/resources/logback.xml` and create a new file called `simplelogger.properties` in the same directory with the following content:

```
# org.slf4j.simpleLogger.defaultLogLevel=debug
org.slf4j.simpleLogger.showDateTime=true
org.slf4j.simpleLogger.dateTimeFormat=yyyy-MM-dd HH:mm:ss:SSS Z
#org.slf4j.simpleLogger.log.my.class.path=DEBUG
org.slf4j.simpleLogger.log.twigkit=DEBUG
```

Going forward, the **simplelogger.properties** file will be used to configure additional logging. The default logging level is `INFO`, but can be changed using the `defaultLogLevel` property. Logging on a particular class can also be increased by adding a new line, such as `org.slf4j.simpleLogger.log.my.class.path=DEBUG`, changing `my.class.path` to the actual path, and setting the logging level as needed.

If your `pom.xml` file contains the following entry, remove it:

```
<dependency> <groupId>ch.qos.logback</groupId> <artifactId>logback-
classic</artifactId> <version>1.1.1</version> </dependency>
```

Improvements

- Swapped out dependencies on Log4j 1.2.x with SL4J in the Appkit code base.

Bug Fixes

- Fix for a bug that would prevent interval facets from toggling. Fix requires the `facet-list` tag to not use the `query=""` attribute.
- Fix for an issue when using the `<widget:spelling-suggestions>` component that would allow a cross-site scripting attack if the query term included a `<script>` tag and the Fusion `spellcheck` component returned a spell correction.

Appkit SDK 4.13.0 Release Notes

Release date: 15 September 2021

	The 4.13.0 release includes items from Appkit SDK 4.12.0, which is not a separate release.

Improvements:

- Upgraded AngularJS framework to 1.8.2
  - Version 1.8.2 meteoric-mining corrects `$sceDelegate` to ensure that `resourceUrlWhitelist()` is identical to `trustedResourceUrlList()`.
- Upgraded JQuery to version 3.60.
  - Version 3.60 corrects security issues during a pen test.

Bug Fixes

- Fixed a bug where the selection of a query:suggestion component added the following to the URL: an `mq` parameter and an `mq value` surrounded by quotes.
- Fixed a bug where the facet filter for a facet already applied did not clear when `select=toggle`.
- Fixed a bug in the search:box with the action attribute where the queries on the search submission page were re-triggered *before* the page was redirected to the value listed in the action attribute.
- Fixed an issue where extra requests were made when a new query was submitted *if* a fixed header and infinite scroll were used.
- Fixed an issue to ensure a facet filter that contains special characters cannot be deselected.

Appkit SDK 4.11.0 Release Notes

Release date: 21 July 2020

Improvements:

- Ability to limit the results displayed by the result-list to a subset of the array
  - For example, the following: `<result-list response="response" first="2" last="5">` will display the 3rd, 4th and 5th results in the response.
- ES7 platform can now be configured whether or not to track total hits.
  - In the ES7 platform config (for example, resources/conf/platforms/elastic/elastic.conf) you can now add this config to track total hits: `track-total-hits: true`
  - By default, this will be **false** and will max out at **10000** as stated in the ES docs.
- It is now possible to specify whether to include or exclude the range filter boundaries by specifying the fromExcluded and toExcluded attributes to the `<query:filter>` tag.
  - For example:

```
<search:query parameters="*" var="query"> <query:filter field="age" from="25" to="50" display="25 to 50" fromExcluded="true" toExcluded="true">
</query:filter> </search:query>
```

- Appkit no longer adds the apollo path prefix to Fusion API URLs. If still required, the path prefix may be set via configuration.

Bug Fixes

- Fixed a bug where `<search:response-statistics>` are only updated when its specified response object is updated, not on any others.
- Added support for Angular expressions as a value for the Bookmark-list `collection` parameter.
- `HighlightingUtils` adds highlighting terms support when closing braces are absent.
- Fusion pre-authentication filter now uses the JWT token subject field for the user name.
- In `collaborate:saved-query-list`, the `show-more` user click option now specifies the number of additional list items to display.

Appkit SDK 4.10.0 Release Notes

Release date: 27 April 2020

Improvements

- A new attribute, `auto-populate`, is added to the `<collaborate:saved-query-list>` tag. When set to `true`, the saved search name is automatically pre-populated with the query term in the URL. The default value for this attribute is `false`.

Bug Fixes

- Fixed a bug that sometimes caused selected filters to be removed from a query after applying new filters on the same facet.
- Fixed a bug that sometimes returned invalid annotation IDs when using the social response processors, which prevented the end-user from further deleting the annotation (like a bookmark or saved-search).
- Specifying how many annotations, such as bookmarks or saved-searches, to retrieve using the Social API would not result in the expected behavior. This has now been fixed.
- Previously, applying the dynamic Solr date range query processor to multiple facets simultaneously in the same configuration resulted in the same gap being applied to all configured facets. This occurred independently of the preexisting filters applied to individual facets, which could lead to inconsistent date range bucketing. This is now fixed.

Appkit SDK 4.9.0 Release Notes

Release date: 10 February 2020

Bug Fixes

- The `FusionProxy` endpoint `/twigkit/api/fusion/*` now accepts nested URI paths as expected. For example:

  ```
  /insights/twigkit/api/fusion/collections/<collection_name>
  /insights/twigkit/api/fusion/apps/<app_name>/experiments
  ```

  Previously, an `HTTP 400` error occurred.
- Removing filters using the `<query:filter action="remove">` tag will no longer remove filters added with the `<query:filter value="bar">` tag. For example, the following will leave the filter on `value="bar"` intact:

  ```
  <search:query var="query"> <query:filter field="foo" value="bar"> <query:filter field="foo" action="remove"> </search:query>
  ```

- Previously, using the Solr platform in compatibility mode forced exclusive boundaries on date range filters to become inclusive. This affected date range filters in the Fusion platform.
  This behavior is fixed. Exclusive boundaries on range filters now remain exclusive in compatibility mode.
- Toggle events now fire as expected when using toggle controls.

Appkit SDK 4.8.0 Release Notes

Release date: 10 January 2020

New features

- It is now possible to map a user's LDAP roles to the application user's group. This differs from the previous implementation to the AD pre-authorization filter, in that it reads the groups directly from the user object instead of querying for AD groups that the user belongs to.
  To enable this filter, add a configuration file under `conf/security/ldap/provider.conf` with the following parameters:

```
enabled: true user: #Service account used to connect to LDAP. password: #Password for the service account. This can be encrypted using the twigcrypt binary. url: #Full URL to the LDAP server. user-search-base: #LDAP search base to restricted the search to. user-search-filter: #LDAP search filter to restrict the LDAP search.
```

Improvements

- It is now possible to use basic authentication against a Solr cloud platform using the parameters `userName` and `password` in the platform configuration.
- The value of a highlighted field is now stored as the "display" field value, rather than replacing the original "actual" field value.
- UTC dates sent from the browser are now retained as UTC when passed to Solr.

Bug Fixes

- All Solr Stats components are now mapped to the Appkit response object. See The Stats Compnents *(https://lucene.apache.org/solr/guide/7_5/the-stats-component.html#statistics-supported)* for more information.
- In provided templates, some of the CSS and JavaScript files under `/twigkit/api/static/**` had the wrong content type. This prevented some browsers from executing the files properly. This is fixed.
- Fixed an issue that sometimes caused the session cookie used for activity tracking to reset and be incorrectly assigned.
- Fixed an issue in the `RoleBasedAuthorisationFilter` that sometimes resulted in a Null Pointer Exception.
- Fixed an issue that sometimes prevented highlighting of multivalue fields.
- Previously, request signals generated by App Studio did not specify whether a filter was excluded or not (a negative filter). Now, filter metadata represents excluded filters with a `-` symbol in front of the filter value.
- Fixed an issue with the `<search:box>` tag that caused the query term used in the search box to differ from the value of the query associated with the tag.

Appkit SDK 4.7.0 Release Notes

Release date: 22 November 2019

New features

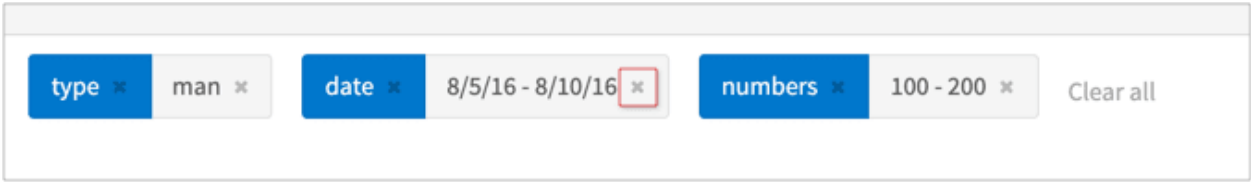
- Appkit 4.7.0 adds support for Elastic 7.4.
- A new filter, `find`, is added which searches the scope hierarchy to find a specified variable. For example: `{{ 'myVariable' | find:this }}`.

Improvements

- The `twigkit.http.filter.DevelopmentErrorFilter` is replaced by the `twigkit.http.filter.ErrorFilter`. If you are using the `twigkit.http.filter.ErrorFilter` parameter in your `web.xml` file and an AJAX request fails, a JSON response is received identifying the status code of the REST call and any underlying causes. Previously, an HTML response was received.
  Example configuration:

```
<filter> <filter-name>ErrorFilter</filter-name> <filter-class>twigkit.http.filter.ErrorFilter</filter-class> <init-param> <param-name>responseCode</param-name> <param-value>400,405,406,500,501,503</param-value> </init-param> <init-param> <param-name>excludedPaths</param-name> <param-value>/,/login,/login/</param-value> </init-param> </filter>
```

- Accessibility improvements are made to the *search breadcrumbs*. The text, `x`, that removed the breadcrumb is replaced by an icon with the alt-text "Remove this term from the query" for use by screen readers.



- For security purposes, any existing incoming `userID` value is removed and replaced when impersonation is enabled.
- When a user enters incorrect login credentials, the username is preserved in the form and the URL for the user to verify.
- The UI now sends a `X-Requested-With=XMLHttpRequest` HTTP header with every API request. This makes it easier for the middleware to detect XHR requests and handle errors appropriately.
- Previously with the `ApplyDynamicSolrDateFormattingResponseProcessor`, the same time format was applied to days, hours, minutes, and seconds. Now, these values have their own formats.
- When using the `twigkit.search.processors.response.ApplyDynamicSolrDateFormattingResponseProcessor` and `twigkit.search.processors.query.DynamicSolrDateRangeQueryProcessor`, a new fact value, `gap`, is added to the date facet. This value of this fact is the unit of the gap size that was applied.

Other changes

- The default session timeout is updated from 10 minutes to 1 hour, to align with Fusion 4.2.x.

Addressed Security Vulnerabilities

- Upgrades to Jackson 2.9.10 *(https://github.com/FasterXML/jackson/wiki/Jackson-Release-2.9.10)* and Apache Commons BeanUtils 1.9.4 *(http://commons.apache.org/proper/commons-beanutils/javadocs/v1.9.4/apidocs/index.html)* are made to address CVE vulnerabilities.

Bug Fixes

- Fixed a bug that occasionally left a filter styled as being selected after another filter was chosen, despite the query results changing to reflect the new selection.
- Fixed an issue that would occasionally let a response object from one view be used by another view.
- Upgraded the JFrog protocol to `https` to prevent a `[FATAL] Non-resolvable parent POM` error.
- Fixed a bug that occasionally caused "like" signals to generate the wrong annotation type metadata.
- Fixed a bug with `FacetFilterValueMappingProcessor` mappings which prevented commas from being used with the value. Additionally, two parameters are introduced, `separator` and `key-value-separator`. For example:

  ```
  separator: ; // Defaults to "," key-value-separator: _ // Defaults to "="
  ```

  Using `key-value-separator: _` will separate the following key value pairs properly: `alpha,beta=greek_a,b=english`.
- Fixed a bug that rarely caused a Null Pointer Exception (NPE) error when a workflow platform was reading an underlying platform parameter.

Appkit SDK 4.6.0 Release Notes

Release date: 27 September 2019

Upgrading to this version

In apps created with earlier versions of Fusion App Studio, you can *upgrade the Appkit version to 4.6.0*.

Additional upgrade steps

If you have previously specified `requestHandler: /` in the platform suggestions configuration files of your apps, you can now remove this parameter.

In earlier versions, `requestHandler: /` was used to allow search term suggestions for apps using a query profile, but this parameter is no longer needed.

For example:

```
# Configure a Platform for Solr Query Suggestions query-profile: YOUR_QUERY_PROFILE
requestHandler: /
```

Upgrading from Appkit versions prior to 4.4

If you are upgrading from a version of Appkit prior to 4.4 then you may need to follow additional steps described in previous Release Notes.

New features

No new features are included in this release.

Improvements

Easier key navigation in result lists

To support easier keyboard navigation in result lists, for example by using the down arrow to move from result to result, Appkit result titles now use HTML heading tags.

Specifically, `<search:field styling="title">` tags now use `<h3>` tags instead of `<p>`.

Bug Fixes

- **Fusion 5 authentication.** Apps created using Appkit 4 and App Studio 4 can now authenticate against Fusion 5.0.
- **Fusion sessions.** Existing Fusion sessions are now correctly utilized when authorizing users against a service account.
- **Term suggestions.** Search term suggestions no longer fail when using a Fusion query profile.
  If you previously used `requestHandler: /` as a workaround for this functionality, carry out the upgrade step described in *Additional upgrade steps* above.
- **Click tracking table results.** The `<track:clicks>` tag now works correctly with `<table:result-list>`.
- **Click tracking fields.** In some conditions, additional fields were incorrectly tracked by the `<track:clicks>` tag. They are now tracked correctly.

Appkit SDK 4.5.0 Release Notes

Release date: 19 August 2019

Upgrading to this version

In apps created with prior versions of Fusion App Studio, you can *upgrade the Appkit version to 4.5.0*.

> Also see the additional required and recommended upgrade steps described in the Release Notes for *Appkit 4.4.0* and *Appkit 4.3.0*.

New features

No new features are included in this release.

Improvements

Role-based authorization now supports multiple roles

Appkit now supports role-based authorization with multiple roles. Previously, it was possible to define only a single, global role-based authorization filter.

Create authorization filters

Create multiple configuration files for role-based access in the `security/access` folder. Each file defines a single authorization filter. When Appkit considers whether a specific user can access a specific resource, Appkit evaluates all of the filters.

> A best practice is to create authorization filters *that do not overlap*, that is, that do not refer to the same resources.

You must give users access to resources explicitly by referencing one or more of their roles. A user who has none of the roles mentioned in the authorization-filter files cannot access any resources.

If a user has multiple roles that specify different access to a resource, then the rule that denies access wins.

Properties

Property	Required	Notes
`allow`	optional	Comma-separated list of roles that allow access to a resource
`deny`	optional	Comma-separated list of roles that deny access to a resource
`pattern`	required	Bar-separated list of paths to which this role-based authorization filter applies

You must supply either an `allow` or a `deny` property. You can supply both, though that is not necessary, because roles that are not allowed are denied. If you do supply both properties and a role is mentioned in both `allow` and `deny`, then `deny` wins.

Example

In this example, users with the roles `EMPLOYEES` or `CONTRACTORS` can access the resources specified in the `pattern` property, whereas users with the role `INTERNS` cannot access the resources.

```
allow: EMPLOYEES,CONTRACTORS deny: INTERNS pattern: (/twigkit/api)|(/dashboard/*)
```

If authorization filters overlap

Although multiple authorization filters can overlap (reference the same resources in the `pattern`), this is not a good practice.

Appkit figures out and applies the most restrictive filter, but results might be counterintuitive. For example, filter 1 might allow users with the role `CONTRACTOR` access to a specific resource, whereas filter 4 denies contractors access to the same resource. Filter 4 wins, but you wouldn't see that if you were looking at filter 1.

Timezone handling

UI components and Appkit server components have been improved to ensure that they handle time zones consistently. Previously, some UI components were aware of local time zones and others were not (instead working with UTC times). The result was that some UI components displayed incorrect times.

Improvements are:

- In the `<search:field>` tag, you can now specify a `timezone` parameter, which tells Appkit the timezone to use when formatting date fields in responses. For example:

  ```
  <search:field name="date" styling="label-left" label="Created Date" date-format="dd/MM/yy hh:mm" timezone="PST"></search:field>
  ```

  Use these time zone abbreviations *(https://en.wikipedia.org/wiki/List_of_time_zone_abbreviations)*.
- Every HTTP request to Appkit's API services includes an `X-Client-Timezone` header, which specifies the browser client's current timezone as a UTC offset *(https://en.wikipedia.org/wiki/List_of_time_zone_abbreviations)*.

Fusion security provider stores JWT to authorize requests to Fusion

Applications that use the Fusion security provider (`twigkit.security.provider.fusion`) can now check whether an incoming request has a JSON web token (JWT) associated with it (in an `Authorization` header that begins with `Bearer`). Appkit extracts the JWT from the header and stores the JWT so that subsequent requests can use the JWT to authorize the requests to Fusion.

The JWT support added in Release 4.5.0 is in addition to the support that was present in prior Appkit releases for pre-authentication with the Fusion security provider.

Now, instead of just checking for a `fusion-user-name` header in the incoming request, Appkit also checks for a JWT header, if it does not find the `fusion-user-name` header.

If a JWT is present, Appkit uses the token to authenticate the user. This is typically used when an application (e.g. App Insights) is running behind a Fusion proxy.

After Appkit retrieves and stores the JWT, Appkit can retrieve the JWT to authorize requests to Fusion. By default, Appkit, handles all of this for you.

Explicit configuration

The means by which a user can authorize a request to Fusion can be made explicit by adding an `auth: auto | serviceaccount | jwt | native` property to the app's Fusion services resources under `resources/services/api/fusion.conf`. The default value of this property is

`native`, which means that Appkit uses the pre-authentication provided by the Fusion security provider.

If you use the `auto` value of the `auth` property, then Appkit works its way through `serviceaccount`, `jwt`, and finally `native`. With the `serviceaccount` option, Appkit tries to find a `userName` and `password` in `resources/services/api/fusion.conf`. With the `jwt` option, Appkit tries to generate a JWT for the user. (This is not the same as the JWT passthrough.)

Geo-filtering and sorting is supported for Elastic 5

In Appkit 4.4.0, support for geo-filtering and sorting *was added to the Elasticsearch search platform*. The support in Appkit 4.4.0 was solely for Elastic 6.

In AppKit 4.5.0, geo-filtering and sorting support has been extended to include Elastic 5.

Accessibility is improved for assistive technologies

Appkit's Angular JS now incorporates ngAria *(https://docs.angularjs.org/guide/accessibility)*, which improves accessibility by including ARIA *(https://www.w3.org/TR/wai-aria/)* attributes in the HTML of Appkit UI components. ARIA attributes help assistive technologies such as screen readers to convey state and semantic information to users.

Bug Fixes

- **Hierarchical facets**. Hierarchical facet requests now always return hierarchical facets. Previously, a `<search:facet-list>` request with multiple hierarchical facets would sometimes return a flattened structure (parent only) for some of the facets.
- **dateFormat and time zone**. The `dateFormat` filter now applies the `timezone` parameter correctly to both date-time values in a date-time range.
- **Responses from multiple suggesters**. When there are multiple suggesters, the `<query:suggestions>` element now includes responses from all suggesters in their respective sections. Previously, only the responses of one suggester were included.

Appkit SDK 4.4.0 Release Notes

Release date: 22 July 2019

Upgrading to this version

In apps created with prior versions of Fusion App Studio, you can *upgrade the Appkit version to 4.4.0*.

> If you are upgrading to 4.4.0 from a version prior to 4.3.0, *additional upgrade steps* are required or recommended.

Recreate topics that contain non-URL-safe characters

An issue in the Social Module in prior releases prevented topics that contained non-URL-safe characters from loading correctly in browsers.

The fix in this release allows *future* topics to contain URL-unsafe characters; the topics load correctly. But this release *does not* repair existing topics.

To repair existing topics, you must delete and recreate the topics.

New features

Elasticsearch search platform supports geo-filtering and sorting

The Elasticsearch search platform now supports geo-filtering and sorting based on distance.

Filter syntaxes

These are the filter syntaxes. The `unit` property is optional. If not present, the default unit `kilometers` is used.

```
f=geo-point-field[{"lat":latitude-value,"lon":longitude-value,"distance":distance-value}][display-value] f=geo-point-field[{"lat":latitude-value,"lon":longitude-value,"distance":distance-value, "unit":"unit-value"}][display-value]
```

Filter properties

Property	Required	Notes
`lat`	required	Latitude value (decimal, for example, `37.773972`)
`lon`	required	Longitude value (decimal, for example, `-122.431297`)
`distance`	required	Distance from a latitude/longitude point (a positive real number, for example, `10` or `2.5`) do not specify the distance units as a part of `distance` (as you would in Elasticsearch). Use `unit` instead.
`unit`	optional	Units of distance. Possible values are the long form distance units *(https://www.elastic.co/guide/en/elasticsearch/reference/6.8/common-options.html#distance-units)*, that is, `miles`, `yards`, `feet`, `inch`, `kilometers`, `meters`, centimeters`, `millimeters`, and `nauticalmiles`. The default is `kilometers`.

Sorting

The syntax of the URL query parameter for sorting on a `geo_point` field is the same as for other sorting (`+` for ascending, `-` for descending):

```
s=+geo-point-field s=-geo-point-field
```

Example queries with geo-filters

Explicit query, default units (kilometers):

```
http://localhost:8080/#/search?
q=*:*&f=pin.location[{"lat":40,"lon":-70,"distance":200}]["Within 200 km"]
```

Implicit query, non-default units (miles):

```
http://localhost:8080/#/search?
f=pin.location[{"lat":40,"lon":-70,"distance":200,"unit":"miles"}]["Within 100
miles"]
```

Implicit query with sorting (descending), default units (kilometers):

```
http://localhost:8080/#/search?f=pin.location[{"lat":40,"lon":-70,"distance":1000}]["Within 1000 km"]&s=+pin.location
```

End-to-end example

In Elasticsearch

1. Create an index. For example:

```
curl -X PUT "localhost:9200/map_locations" -H 'Content-Type: application/json' -d' { "settings" : { "number_of_shards" : 3, "number_of_replicas" : 2 } } '
```

2. Apply a geo-point mapping to a field in the index. For example:

```
curl -X PUT "localhost:9200/map_locations" -H 'Content-Type: application/json' -d' { "mappings": { "_doc": { "properties": { "pin": { "properties": { "location": { "type": "geo_point" } } } } } } } '
```

3. Index documents that contain one or more fields of type `geo_point` *(https://www.elastic.co/guide/en/elasticsearch/reference/6.8/geo-point.html)* (any of the four formats). For example:

```
curl -X PUT "localhost:9200/my_locations/_doc/1" -H 'Content-Type: application/json' -d' { "pin" : { "location" : { "lat" : 40.12, "lon" : -71.34 } } } ' curl -X PUT "localhost:9200/my_locations/_doc/2" -H 'Content-Type: application/json' -d' { "pin" : { "location" : { "lat" : 42.12, "lon" : -74.34 } } } ' curl -X PUT "localhost:9200/my_locations/_doc/3" -H 'Content-Type: application/json' -d' { "pin" : { "location" : { "lat" : 0.12, "lon" : -1.34 } } } '
```

4. In the Elasticsearch configuration file *(https://www.elastic.co/guide/en/elasticsearch/reference/6.8/settings.html)* `elasticsearch.yml`, identify the field or fields that are of type `geo_point`. For example:

```
geo-point-fields: pin.location
```

In Appkit

1. Perform a search with a filter that identifies a `geo_point` field and specifies a latitude, a longitude, and a distance (and optionally units). For example:

```
http://localhost:8080/{hash}/search?
q=*:*&f=pin.location[{"lat":40,"lon":-70,"distance":200}]["Within 200 km"]
```

2. Optionally, include an `s` (sort) query parameter to sort search results on a `geo_point` field that was used in a filter; for example, `s=+pin.location` (ascending) or `s=-pin.location` (descending). For example:

```
http://localhost:8080/{hash}/search?
q=*:*&f=pin.location[{"lat":40,"lon":-70,"distance":200}]["Within 200
km"]&s=+pin.location
```

3. The Elasticsearch platform performs an Elasticsearch `geo_distance` query, which finds documents with at least one geo-point that falls within the specified distance of the specified point (and that satisfy any other query and filter criteria), and returns a response that lists the documents. If the `s` query parameter was included, the documents listed in the response are sorted in order of distance from the specified point.

Improvements

Appkit filter syntax supports complex filter values

Previously, the Appkit filter syntax supported a small number of simple data types (`string`, `boolean`, `integer`, `double`, and `date`) and ranges over simple data types.

With this release, the Appkit filter syntax adds support for an `object` data type that can specify filters of arbitrary complexity. The `object` data type uses JSON syntax *(https://json.org/)* to describe the data. For example, a filter can specify unordered lists of name/value pairs (objects) and ordered collections of values (arrays).

For example:

```
data[{"property1":"some_text","property2":34,"property3":true,"property4":922337203
6854775807,"property5":53.4678,"property6":["a","b"],"property7":
{"key1":"value","key2":56}}]["A filter object"]
```

Security providers return 401 Unauthorized for unauthenticated API requests

All Appkit security providers that use Spring security now return a `401 Unauthorized` status for unauthenticated API requests. Previously, they returned a `403 Forbidden` status.

Additionally, for the SAML security provider, the standard AJAX request handling mechanism has been included, which means that `302 Found` redirects do not occur for unauthenticated API requests.

The security providers affected are:

- Active Directory
- Twigkit Cookie Cracker
- Lucidworks Fusion Sessions API
- Kerberos
- LDAP
- ADFS OAuth
- Facebook OAuth

- Google Oauth
- Office 365 OAuth
- SAML
- Spring Security

Histograms and sliders work with small timeframes

Histograms and sliders now work correctly with date ranges that have small timeframes, for example hours or minutes.

The filter-optional attribute has been added to <search:field>

When using `<search:field>` to make a field clickable in order to refine the current query, you can now define the `filter-optional` tag attribute. This tag attribute tells Appkit how to combine filters using connective logical operators.

For example, when you have a multi-select (checkboxes) facet defined on the same field and displayed on the same page as a clickable field for refining a query, you must specify `filter-optional` to be `value` in the filter for the facet directive.

For more information, see *Optional logical operators*.

Bug Fixes

- Click tracking for instant results (`<instant:result-list>`) now returns the correct `query` object in the tracked-click object. These are examples of markup and a response object:

```
<search:box placeholder="Find documents..." action="search" social="true"
query="query"> <track:clicks query="query"> <!-- Instant Results --> <div>
<instant:result-list platform="platform" query="query"> <search:result>
<search:field name="title" styling="title" urlfield="url"></search:field>
</search:result> </instant:result-list> </div> </track:clicks> </search:box>
```

```
{ id: "quickstart/Revolution-Session-Data.csv#36", label: "Passage Search: the
Answer to Your User Questions!", offset: 0, pos: 3, query: {rpp: "14", fa:
"keywords_s", p: 1, q: "sa*"}, type: "result", url:
"http://localhost:8090/quickstart/Revolution-Session-Data.csv#36" }
```

- When using the Social module, topics that contain non-URL-safe characters (for example, `#`) now load in browsers correctly, *if the comments were created using this software version, 4.4.0*.

  > This release does not repair existing topics that contain non-URL-safe characters. You must *recreate those comments*.

- When using the `<search:field>` tag, fields with no value are no longer clickable.
- The localization key `collaborate.saved.placeholder` was added for the hint text in the search box for the `collaborate:saved-query-list` tag. The search box lets the user search through a list of saved queries. The default English hint text is `Search name`.
- The root context page is now an allowed login location.
- When grouping results, the first related result no longer shows values for all related results.
- When grouping results using the platform `groupField` parameter, the first result in a group of results now contains only its own metadata. Previously, the first result contained the metadata of all of the results in the group.

- The `<widget:date-range>` tag now completely clears dates when the date filter is cleared.

Appkit SDK 4.3.0 Release Notes

Release date: 29 May 2019

Upgrading to this version

In apps created with prior versions of Fusion App Studio, you can *upgrade the Appkit version to 4.3.0*.

This section explains *additional steps* that are required or recommended (as indicated) for upgrading to this version of Appkit.

Spring Security XML schema

Update your apps that use Spring security to use version 4.2 of the Spring Security XML schema. In any application that has its own `spring-security.xml` file:

1. Replace this Spring Security XML schema declaration:

   ```
   http://www.springframework.org/schema/security/spring-security-4.1.xsd
   ```

   With this one:

   ```
   http://www.springframework.org/schema/security/spring-security-4.2.xsd
   ```

Using a Fusion service account for authentication and authorization

> If you used a service account to authenticate against Fusion in a prior Appkit version, you must move the `userName` and `password` configuration values from your Fusion platform configuration to `services/api/fusion.conf` in this version.

Service accounts no longer require the `impersonate` configuration option to be set to `true`. If the option is set to `true` in `services/api/fusion.conf`, it is ignored.

For more information, see *Fusion authorization*.

Authorization and pre-authorization filters

For existing applications that implement custom `AuthorizationFilter` logic, you can and should re-implement those filters as instances of `PreAuthorizationFilter`, if the intention of those filters is not to authorize a request, but instead to augment the user object with pre-authorization information. However, this is not a necessary requirement for upgrading, because existing authorization filters work exactly as before. (They run after the pre-authorization filters.)

Changing how to send signals to Fusion

> This change was introduced in Appkit 4.0.0. If you have not already made this change, it is necessary now because the old approach to generating signals is *deprecated and has been removed*.

How to change how signals are sent to Fusion

1. Remove the following from your `pom.xml` file:

```
<dependency> <groupId>twigkit</groupId> <artifactId>twigkit.activity.fusion-signals</artifactId> <version>${project.parent.version}</version> </dependency>
```

2. Add the following to your `pom.xml` file:

```
<dependency> <groupId>twigkit</groupId>
<artifactId>twigkit.activity.tracker</artifactId>
<version>${project.parent.version}</version> </dependency> <dependency>
<groupId>twigkit</groupId>
<artifactId>twigkit.message.service.fusion.producer</artifactId>
<version>${project.parent.version}</version> </dependency>
```

3. Add configuration parameters in `conf/message/service/fusion.conf`. You can send signal events to Fusion using either the Signals API or the Query API. This is an example in which the query-profile parameter is used to send events with the Fusion Query API:

```
query-profile: my-profile commit: true async: false signals-index-pipeline: my-index-pipeline
```

   The parameter `signals-index-pipeline` is an optional parameter that defines a specific index pipeline to use when indexing signals. If no index pipeline is specified, the pre-configured `_signals_ingest` pipeline is used.

The `signals-endpoint` configuration parameter is no longer supported.

4. By default, Appkit sends all events to the Signals endpoint. If you want to customize which events are sent, add a configuration file at `activity/tracking/tracker.conf` with the property `events`:

```
events: click, request, response, login, annotation # or any subset thereof
```

A query pipeline automatically generates response signals when the signals feature is enabled for a collection. The search application should not send response signals to Fusion directly, because those would conflict with the automatically generated signals.

New features

- **JWT authorization against Fusion is supported**. Appkit now supports JSON Web Token (JWT) authorization against a Fusion JWT realm.
  For information about options for authorization in Fusion, see *Fusion authorization*.
- **Discovery of Fusion proxy instances**. By default, Appkit now attempts to discover a Fusion Proxy service instance for querying. This removes the need to colocate a Fusion Proxy Service instance on every node running the Fusion Web Apps service.
  - If Appkit discovers a proxy, Appkit uses the discovered proxy instead of the configured `host` and `port` in `services/api/fusion.conf`.
  - If Appkit does not discover a proxy, Appkit uses the configured `host` and `port` in `services/api/fusion.conf`, or `localhost:6764` if the configuration values are not set.

> Proxy discovery uses the `fusion-proxy` Java system property, which is usually only available when an Appkit app runs on the Fusion Web Apps service.

Configuration:

To specify how Appkit finds a Fusion Proxy Service instance, define the property `fusion-discovery-enabled` in `services/api/fusion.conf`:

- `fusion-discovery-enabled: true` (default if omitted). Appkit attempts to discover a Fusion proxy instance by reading the `fusion-proxy` Java system property. If it does not discover a proxy, Appkit uses the configured `host` and `port` in `services/api/fusion.conf`, or `localhost:6764` if the configuration values are not set.
- `fusion-discovery-enabled: false`. Appkit does not attempt to discover a proxy, and uses the configured `host` and `port` in `services/api/fusion.conf`, or `localhost:6764` if the configuration values are not set.

  > In an existing application that is deployed to Fusion, but that is not designed to query the Fusion on which it is deployed for the location of the Fusion Proxy service, you must add the `fusion-discovery-enabled` parameter to `fusion.conf` with a value of `false` to retain the current behavior.

- **Role-based authorization**. Appkit adds an authorization filter that can authorize requests for protected resources based on the roles associated with the user who made the request. Security providers (such as SAML) or LDAP lookups can provide the roles. For information, see *Role-based authorization*.
- **Solr JSON range facets**. Appkit now supports Solr JSON range facets as follows:
  - Appkit can now parse facets that were generated using JSON syntax and leverage them for display purposes in Appkit UI components.
  - Appkit cannot generate JSON facet query syntax (which needs to be manually input in the query pipeline or as query custom parameters).
  - Appkit cannot use the JSON facet's filters to further refine query.
- **Localization**. Appkit replaces hard-coded labels (in English) with calls to an Appkit translation service. The service provides a label in the chosen language, with a fallback to English if no translation exists for the label in the chosen language. For more information, see *Localization service*.

  > This changes the *attributes for a number of tags*.

- **Stack traces in browser**. Appkit can send stack traces to the browser in responses (instead of HTML error pages) when run in development mode.
  To enable full stack traces:

  1. Add a new filter, `twigkit.http.filter.DevelopmentErrorFilter`:

```
<filter> <filter-name>ErrorFilter</filter-name> <filter-
class>twigkit.http.filter.DevelopmentErrorFilter</filter-class> <!-- <init-param>--
> <!-- <param-name>responseCode</param-name>--> <!-- <param-value>500</param-
value>--> <!-- </init-param>--> </filter> <filter-mapping> <filter-
name>ErrorFilter</filter-name> <url-pattern>*</url-pattern>
<dispatcher>ERROR</dispatcher> </filter-mapping>
```

  1. To only send stack traces for a subset of error response codes, e.g. 500 and 404, uncomment the `<init-param>` block and specify the response codes as a list of comma-separated values in `<param-value>`. For example:

```
<init-param> <param-name>responseCode</param-name> <param-
value>403,404,500</param-value> </init-param>
```

Improvements

- **Retrieval of properties by workflow platforms.** If a platform is configured to be a workflow platform, platform property retrieval proceeds as follows:
  a. Appkit checks to see whether the workflow platform has a property.
  b. If the workflow platform does not have the property, Appkit checks to see whether the underlying platform has the property.
  c. If both the workflow platform and the underlying platform have the same property, the one defined for the workflow takes precedence.
- **Solr result clustering.** Appkit adds support for Solr result clustering *(https://lucene.apache.org/solr/guide/7_1/result-clustering.html)*.
- **No longer necessary to configure wt=xml in Fusion.** Appkit's Solr Suggestions service that is used when backward compatibility is enabled (`SolrSuggestionsParserService`) now appends `wt=xml` to the Solr suggestions URL automatically. You no longer need to configure this query parameter in the Fusion query profile for suggestions.
- **Improved performance in Social module.** Performance of the Social module has been improved for annotation deletions.
- **Overwrite context path.** It is now possible to overwrite the context path. This is necessary when search app HTML files are hosted on one server and the Appkit APIs are on a different server.
- **No separate license restriction for Workflow module.** The separate license restriction for the Workflow module has been removed.
- **Lucene version.** The `twigkit.lucene` module now uses Lucene v7.2.1. You can use the `twigkit.lucene` module with other modules, for example, Elasticsearch 6, that use more recent versions of Lucene.
- **Default requestMethod for Solr search platform.** The default `requestMethod` for the Solr search platform has been changed from GET to POST.

Role-based authorization

In many cases, you might want to restrict access to your application, or parts of your application, to specific groups of users. To enable this option, you must add the file `conf/security/access.conf` to your application's configuration tree with these configuration parameters:

```
allow: role-with-access-1,role-with-access-2,... deny: role-without-access-1,role-without-access-2,... pattern: regex-of-uris-to-intercept
```

These configuration parameters are:

- `allow`: A comma-separated list of roles that should be granted access. When not specified or set to a wildcard ('*'), Appkit defaults to allowing access to all roles.
- `deny`: A comma-separated list of roles that should be denied access.
- `pattern`: A regular expression that defines the pattern of URIs that should be intercepted for checking authorization. This defaults to .* (all paths are checked).

Example

Let us assume that you have a user directory containing two users, `user` and `admin`, where only the latter has an `ADMIN` role. For example, you can define this using a simple `spring-security.xml` configuration with a static list of users, like this:

```
<authentication-manager> <authentication-provider> <user-service> <user name="user" password="user" authorities="USER"/> <user name="admin" password="admin" authorities="USER,ADMIN"/> </user-service> </authentication-provider> </authentication-manager>
```

To configure role-based access, add `conf/security/access.conf` to the application with this configuration:

```
allow: ADMIN pattern: (/)|(/twigkit/api/.*)
```

This says that, for all requests to `/` (the root page) and the API service, Appkit applies role-based authorization rules. Including the root page is not strictly necessary, but it leads to a better experience for users who are denied access.

To validate this setup, first log in as `user`. Appkit denies you access, as shown in the application logs:

```
TRACE t.s.SecurityFilterExecutionModule - Filtering with [twigkit.security.filter.RoleBasedAuthorizationFilter] TRACE t.s.f.InterceptAuthorizationFilter - Filtering request to / - comparing against access pattern (/)|(/twigkit/api/.*) TRACE t.s.f.RoleBasedAuthorizationFilter - User 'user' is DENIED access to protected resource ERROR t.s.SecurityFilterExecutionModule - Authorization chain failed - returning 403
```

Next, log in as `admin`. Appkit allows you to access the app, as shown in the application logs:

```
TRACE t.s.SecurityFilterExecutionModule - Filtering with [twigkit.security.filter.RoleBasedAuthorizationFilter] TRACE t.s.f.InterceptAuthorizationFilter - Filtering request to / - comparing against access pattern (/)|(/twigkit/api/.*) TRACE t.s.f.RoleBasedAuthorizationFilter - User 'admin' is GRANTED access to protected resource
```

Bug Fixes

- The default `requestMethod` for the Solr search platform has been changed to POST to avoid URL length limitations. Previously, the default was GET.
- An Apache Xerces dependency conflict between the SAML and Fusion security providers has been resolved.
- An error was corrected in the Spring security file `spring-security.xml` for Active Directory authentication. The `<security:loginForm>` tag lacked this line, which has been added:

```
login-processing-url="/j_spring_security_check"
```

- When using the Social module with the Fusion platform, the index pipeline and query pipeline now default to `COLLECTION_NAME_system` if `social.conf` specifies the `collection`.
  Instead of specifying the `collection` in `social.conf`, you can specify the `index-profile` and `query-profile`.
- The `slider` widget now submits the correct query.
- The bookmark icon on a result document now correctly indicates whether the document is bookmarked. Previously, deleting a bookmark from the bookmark list or search-box dropdown did not update the icon on the document.
- Click signals are now posted correctly to Fusion when Fusion is configured for single sign-on (SSO).
- The CSRF profile filter occasionally returns the wrong request tokens for a user.
- Previously, some query strings were incorrectly URL-encoded when being persisted in the Social module. The query strings are now persisted correctly.
- The `<search:facet>` tag now displays breadcrumb names such as `date` that were erroneously omitted.
- In the Social module, likes are now deleted when the user deselects the Like (thumb) icon.
- The task framework now allows completed tasks to be deleted.
- When using Appkit Guice security filters (not Spring security) failed authentication requests now return 401 rather than 200.
- When `label-field` is not set, Appkit no longer displays an empty tooltip when hovering over a marker on a map.

New features

The following are new features in this release:

Deprecations

The following are deprecations in this release:

- **Old method of sending signals to Fusion**. How to send signals to Fusion has changed in this release. The `twigkit.activity.fusion-signals` module is deprecated and has been removed.
  Use the `twigkit.message.service.fusion.producer` module instead. For more information, see *Changing how to send signals to Fusion*.
- **Service account configuration option 'impersonate'**. Service accounts no longer require the `impersonate` configuration option to be set to `true`. If the option is set to `true` in `services/api/fusion.conf`, it is ignored.

An *alternative means of authorizing with a service account* is provided.

	If you used a service account to authenticate against Fusion in a prior Appkit version, you must move the `userName` and `password` configuration values from your Fusion platform configuration to `services/api/fusion.conf` in this version.

Improvements

The following are improvements in this release:

Fusion authorization

Appkit now has three Fusion authorization modes:

Mode	Description
Fusion native realm	Authorization is against the *Fusion native security realm* directly, using Fusion session passthrough. This is most useful for prototyping apps with Fusion's Web Apps service.
Service account	Authorization uses a service account in the Fusion native security realm. This is useful with external security providers, but entails having a username and password in the Appkit configuration.
Fusion JWT realm	Authorization is of individual users against the *Fusion JWT security realm*. This is also useful with external security providers, and there is no username and password in the Appkit configuration that might allow an attacker to gain access to the Fusion UI.

Specify one of these modes or `auto` in the *Fusion authorization configuration*.

Service accounts no longer require the `impersonate` configuration option to be set to `true`. If the option is set to `true` in `services/api/fusion.conf`, it is ignored. An *alternative means of authorizing with a service account* is provided.

> If a configured authorization mode is not available (that is, configuration keys are missing), then Appkit uses the `native` mode as a fallback. An application is never left in a state in which it cannot at least try to authorize.

Configuration

By default, Appkit reads `services/api/fusion.conf` for configuration.

Fusion configuration

Provide the `auth` configuration parameter in `services/api/fusion.conf` to configure Fusion authorization. The `auth` parameter can have these values.

Value	Description
`native`	Authorization is against the Fusion native realm directly, using Fusion session passthrough. This is most useful for prototyping apps with Fusion's Web Apps service.
`serviceaccount`	Authorization uses a service account in the *Fusion native security realm*. This is useful with external security providers, but entails having a username and password in the Appkit configuration.
`jwt`	Authorization is of individual users against the *Fusion JWT security realm*. This is also useful with external security providers, and there is no username and password in the Appkit configuration that might allow an attacker to gain access to the Fusion UI.
`auto` (default)	Appkit searches for an available authentication provider in the order (1) `jwt`, (2) `serviceaccount`, and (3) `native`. Appkit uses the first authorization provider that confirms it has the necessary configuration to attempt authorization.

Service account configuration

To configure Appkit for service account authorization against Fusion, include the following configuration options in `services/api/fusion.conf`:

Option	Description
`userName`	User name of the user for authorization (String, encryptable)
`password`	User’s password (String, encryptable)

Service accounts no longer require the `impersonate` configuration option to be set to `true`. If the option is set to `true` in `services/api/fusion.conf`, it is ignored.

> If you used a service account to authenticate against Fusion in a prior Appkit version, you must move the `userName` and `password` configuration values from your Fusion platform configuration to `services/api/fusion.conf` in this version.

JWT configuration

To configure Appkit for JWT authorization against Fusion, include the following configuration options in `services/api/fusion.conf`:

Option	Description
`issuer`	Issuer that is configured in the Fusion JWT security realm (String, encryptable)
`key`	Key that is configured in the Fusion JWT security realm (String, encryptable)
`groups`	Group or list of groups. Groups must match those configured in the Fusion JWT security realm (Comma-separated values, encryptable)
`subject`	Override the current user’s ID with this value. Use this when all JWT requests should be sent as a single user (for impersonation or service account emulation). (String; optional, encryptable)

Native configuration

No configuration is required for `native` authorization.

App localization

You can localize the UI labels in an app in multiple languages of your choice.

Localization service

Appkit obtains the correct labels for a language by making calls to an Appkit translation service. The service provides a label in the chosen language, with a fallback to English if no translation exists for the label in the chosen language.

Loading translations

You can use the new `<translations:localize>` JSP tag to load a set of UI label translations into an app. The tag syntax is:

```
<translations:localize dictionary="*dictionary*" locale="*locale*" />
```

- **dictionary**. Name of the resource bundle from which to load translations. This is also the first part of the name of each property file that contains translations.
  Default: translations
- **locale**. Locale
  Default: en (English)

The ***dictionary*** and ***locale*** are used to find the file as follows:

1. In src/main/resources, the Appkit translation service looks for a resource bundle named ***dictionary***, for example, translations.
2. Within the resource bundle, the translation service looks for property files named ***dictionary_locale***.properties, for example, for the files translations_en.properties and translations_fr.properties.

Enabling translations

1. Find label strings that need to be translated in the UI and identify the *key names for the labels*.
2. Create translation property files and add them to a resource bundle in src/main/resources. For example, add translations_en.properties and translations_fr.properties to the resource bundle src/main/resources/translations.
   Edit the property files and include the label keys and translations. For example:
   In translations_en.properties:

   ```
   # translations_en.properties ... components.pagination.next = Next
   components.pagination.previous = Previous ...
   ```

   In translations_fr.properties:

   ```
   # translations_fr.properties ... components.pagination.next = Suivant
   components.pagination.previous = Précédent ...
   ```

   > Some translations include variables. For example, components.response-statistics.showing is Showing {first} - {last} of {total} in English. Translations should retain the variables, though their order might differ. Labels can use tags for emphasis, for example, More results like {result}.
3. Add this taglib directive at the top of your app's index.jsp:

   ```
   <%@ taglib prefix="translations" uri="/twigkit/translations" %>
   ```

4. Add the Appkit <translations:localize> JSP tag inside the <head> element of your app's index.jsp:

   ```
   <translations:localize locale="${param.locale}" />
   ```

5. Add this rule to the url-rules.xml file:

```
<!-- Language specific pages --> <rule> <from>^/lang/(.*)/</from> <to
last="true">/index.jsp?locale=$1</to> </rule>
```

> You can modify other URL rules to suit your application logic. The
> `<translations:localize> JSP tag works the same.

5. Restart your app.
7. Verify that the translations are present by visiting pages in the app that should be in different languages.

Tag attribute changes

There are localization-related attribute changes for these tags:

- `<collaborate:bookmark>`: Attributes `tooltip`, `active-tooltip`, `singular-tooltip`, and `plural-tooltip` have been removed.
- `<collaborate:bookmark-list>`: Attribute `empty-text` has been removed.
- `<collaborate:comment-list>`: Attributes `delete-label`, `show-more-label`, `show-less-label`, and `default-delete-message` have been removed.
- `<collaborate:like>`: Attributes `tooltip`, `active-tooltip`, `singular-tooltip`, and `plural-tooltip` have been removed.
- `<collaborate:topic-list>`: Attribute `empty-text` has been removed.
- `<search:breadcrumbs>`: Attribute `exclude-label` has been removed.
- `<search:facet>`: Attributes `show-more-label` and `show-less-label` have been removed.
- `<search:pagination>`: Attributes `next-label` and `previous-label` have been removed.
- `<search:response-statistics>`: Attributes `showing-all-label`, `showing-label`, `of-label`, and `to-label` have been removed.
- `<widget:spelling-suggestions>`: Attributes `label`, `original-label`, and `corrected-label` have been removed.

Appkit SDK 4.2.5 Release Notes

Release date: 5 April 2019

Upgrading to this version

In apps created with prior versions of Fusion App Studio, you can *upgrade the Appkit version to 4.2.5*.

New features and improvements

There are no new features or improvements in this release.

Fixed issue

The following issue is fixed in this release:

- Click tracking result position does not work with nested results.

Appkit SDK 4.2.4 Release Notes

Release date: 5 April 2019

Upgrading to this version

In apps created with prior versions of Fusion App Studio, you can *upgrade the Appkit version to 4.2.4*.

New features and improvements

There are no new features or improvements in this release.

Fixed issues

The following issues are fixed in this release:

- Spring security configuration for Active Directory is missing `login-processing-url`.
- Breadcrumbs do not display the field name "date".
- Likes are not deleted when a user unselects the thumb icon.
- Highlighting in table cells displays emphasis as plain-text `<em></em>` tags.

Appkit SDK 4.2.3 Release Notes

Release date: 25 March 2019

Upgrading to this version

In apps created with prior versions of Fusion App Studio, you can *upgrade the Appkit version to 4.2.3*.

New features and improvements

You can add the optional `search-wildcard` attribute to the `search-facet` directive to specify whether Appkit performs a wildcard search within facet names and, if it does, how it attaches the wildcard character(s) to the terms that users enter.

A wildcard character matches zero or more characters where it is placed. For example, `computer*` would match the facets `computers` and `computer accessories`. This wildcard is often represented as an asterisk (`*`).

The `search-wildcard` attribute can have these values:

Value	Description
`left`	Adds a wildcard prefix to facet search terms, for example, `term` (entered) becomes `*term` (used to search).
`right`	Adds a wildcard suffix to facet search terms, for example, `term` (entered) becomes `term*` (used to search).
`both` (default)	Adds both a wildcard prefix and suffix to facet search terms, for example, `term` (entered) becomes `*term*` (used to search).
`none`	Adds no wildcard to facet search terms (exact match).

Fixed issues

This release contains these fixes:

- Previously, some click signals caused HTTP 401 errors. These are now sent correctly.
- Saved queries with page parameters now work correctly.

Appkit SDK 4.2.2 Release Notes

Release date: 14 March 2019

Upgrading to this version

In apps created with prior versions of Fusion App Studio, you can *upgrade the Appkit version to 4.2.2*.

New features and improvements

There are no new features or improvements in this release.

Fixed issues

The following issues are fixed in this release:

- The Cross-Site Request Forgery (CSRF) profile filter occasionally returned the wrong request tokens to the user.
- After deleting a bookmark from a bookmark list, the bookmark icon on the result document remained in the "bookmarked" state.

Appkit SDK 4.2.1 Release Notes

Release date: 8 March 2019

Upgrading to this version

In apps created with prior versions of Fusion App Studio, you can *upgrade the Appkit version to 4.2.1*.

New features and improvements

There are no new features or improvements in this release.

Fixed issues

The following issue is fixed in this release:

- Using a two-date time picker widget, a user could select a From time that was after the To time. The `widget:date-range` element now ensures that the user can only select a From time that precedes the To time.

Appkit SDK 4.2.0 Release Notes

Release date: 28 February 2019

Appkit SDK 4.2.0 is the version of Appkit that is included in Fusion App Studio 4.2.0.

Also see the *App Studio 4.2.0 release notes* and the *Fusion 4.2.0 release notes*.

Upgrading to this version

Appkit 4.2.0 is the version of Appkit that is included in Fusion App Studio 4.2.0.

In apps created with prior versions of Fusion App Studio, you can *upgrade the Appkit version to 4.2.0*.

New features

The following are new features in this release.

Social module

- **Single response processor for social module.** You can now configure a single response processor for the Social module, instead of having to configure separate ones for likes, comments, bookmarks, and so forth.
- **Persist social data in Fusion.** You can now *persist Social module data in a Fusion collection*.

Fusion

- **Query against the Fusion Analytics Catalog.** A Fusion Analytics Catalog platform has been added to query against the Fusion Analytics Catalog collection. This platform translates Appkit queries into SQL statements which are executed against a Fusion query pipeline that includes an *Analytics Catalog Stage*. The Fusion Analytics Catalog platform uses the *Fusion Catalog API*.
  For more information, see the *Fusion Analytics Catalog platform*.
- **Fusion platform supports index profiles.** The Fusion platform adds the ability to use an index profile to index data into Fusion. When a read/write Fusion platform is present and an index profile is specified, then Appkit can store, update, and delete data in the Fusion index.
  To use an index profile, put `index-profile: ***my-index-profile-name***` in your Fusion platform configuration, replacing ***`my-index-profile-name`*** with the index profile Appkit will use. Index profiles take precedence over index pipelines.
- **Use Fusion preauthentication.** Fusion preauthentication is now available in the Fusion Security Provider module. By default, if an app is detected as having been uploaded to Fusion and the user has already logged into Fusion, then the user will not need to log into the app.

Miscellaneous

- **Render facets in tabular format.** A new web service lets you render facets in a tabular format.
  A new API endpoint, `/core/services/platform/table`, lets you retrieve the output of a Fusion/Solr pivot query.
  The endpoint renders the data as a set of headers and a set of rows. The output looks like this:

  ```
  {headers: [h1,h2,h3,h4]; rows: [c11,c12,c13,c14], [c21,c22,c23,c24] }
  ```

  You can leverage the endpoint in `pivot:table` elements.

- **Satellite platform can delegate search requests.** A Satellite Platform can delegate search requests to any underlying platform.
  To use a satellite platform, place a new platform configuration file under the app's resources; for example, a `satellite.conf` configuration file in `conf/platforms/satellite/`. Set these parameters in your platform configuration file:
  - `name`: `twigkit.search.satellite.Satellite` (required)
  - `host`: URL of the host for the underlying platform. It must begin with `http` or `https`.
  - `platform-config`: Platform configuration being delegated to
  - `username`: Username for authentication on the underlying platform
  - `password`: Password for authentication on the underlying platform. It is the encoded output of the Appkit `twigcrypt` utility.

  The `username` and `password` are required if Basic authentication is required to request search results from the underlying platform, for example, as is the case with Solr.
  For example:

```
name: twigkit.search.satellite.Satellite platform-config: platforms.fusion.data
username: search-user password: Enc(hfuqeif894732)
```

- **Query parameters are included in tab links.** Query parameters are now included automatically in the links for tab names.
- **Query by drawing rectangles on ESRI maps.** Users of Appkit web apps that display ESRI maps can now make geospatial queries by drawing rectangles on maps. The control for drawing rectangles is between the plus and minus signs.
  Google Maps has had this feature previously.
- **Specify date formats.** The Date Format processor `DateFormatProcessor` lets you specify the format for the display value of a date field.
  For more information, see *DateFormatProcessor*.
- **Use JSP tag library utility functions.** A JSP tag library and utility functions within the library are available to JSP pages.
  A dependency on `twigkit.fusion.ui-jsp` provides a JSP tag library. Adding `<%@ taglib prefix="app" uri="/lucidworks/app" %>` to the top of a JSP page exposes utility functions within that library. Adding `${app:contextPath(pageContext.request)}` sets the context path of the app on that page, taking into account the Fusion webapps proxy.
- **Use Elasticsearch 6 spell correction/did-you-mean.** Support has been added for Elasticsearch 6 spell correction/did-you-mean functionality via term and phrase suggesters:
  https://www.elastic.co/guide/en/elasticsearch/reference/6.3/search-suggesters-term.html *(https://www.elastic.co/guide/en/elasticsearch/reference/6.3/search-suggesters-term.html)*
  https://www.elastic.co/guide/en/elasticsearch/reference/6.3/search-suggesters-phrase.html *(https://www.elastic.co/guide/en/elasticsearch/reference/6.3/search-suggesters-phrase.html)*

Improvements

The following are improvements in existing features in this release.

- **Date formats.** Date formats are applied to date ranges in facet series in charts.
- **Range values.** The `numberFormat` and `dateFormat` value filters now handle range values automatically by formatting the endpoints of a range individually. With this change, you must remove any special handling of range values in individual widgets.
- **Multiselection of highmaps regions.** A new `select` attribute has been added to the `geo:highmaps` tag, which takes these values:
  - `click` (default) - User selects a filter by clicking a region.
  - `multiselect`.* Select multiple filters that get combined with a logical disjunction (OR). Shows an Apply button that is enabled if and only if one or more checkboxes

have been ticked.
    - `multiselect-instant`. Select multiple filters that get combined with a logical disjunction (OR). Filters are applied immediately when a checkbox is ticked.
- **Tag search:dropdown renamed**. The `search:dropdown` tag has been replaced with the `facet:dropdown` tag. All functionality from the prior tag exists in the new one.
- **Control filter selection for facet:dropdown**. A new `select` parameter has been added to the `facet:dropdown` tag, which takes these values:
    - `toggle` (default) - Only allows one filter to be selected at a time, but shows unselected filters and counts.
    - `multiselect`.* Select multiple filters that get combined with a logical disjunction (OR). Shows an Apply button that is enabled if and only if one or more checkboxes have been ticked.
    - `multiselect-instant`. Select multiple filters that get combined with a logical disjunction (OR). Filters are applied immediately when a checkbox is ticked.
- **Pass options to geo:highmaps**. The new `options` parameter lets you pass other options to `geo:highmaps`.
- **Show more values for search:filter**. A new `show-more-values` parameter has been added to the `search:filter` tag.
    - When `show-more-values` is specified without an assignment (e.g. `<search-field show-more-values …>`) or with the value `-1`, then Appkit shows all additional values for the field when the user clicks the "Show more" link.
    - When `show-more-values` is assigned a positive integer value (e.g. `<search-field show="5" show-more-values="10" …>`), then Appkit shows that number of additional values of the field when the user clicks the "Show more" link (or all remaining values if there are fewer values than the number requested).
    - In both cases, Appkit does not show a "Show more" link when there are no additional values to be shown.
      Example usage

      ```
      <search:result-list results="results"> <search:result> <search:field name="multivalued-1" show="2" show-more-values></search:field> <search:field name="multivalued-2" show="2" show-more-values="10"></search:field> </search:result> </search:result-list>
      ```

- **New parameters for facet:dropdown**. A number of additional parameters are now available in the `facet:dropdown` tag:

  ```
  facet-name title show show-more max-characters select show-empty show-more-label show-less-label show-count action order count-number date-format measure
  ```

  These parameters were previously available in the `search:filter-list` tag. They have the same uses.
- **Track clicks for any clickable elements**. You can add click tracking to any clickable element on a page, including clicks on elements that are not anchor elements (and not even hyperlinks). To configure click tracking on custom elements, include your block of trackable elements in a `<track-clicks>` element, as before, and add one or more of the following attributes to the DOM elements for which you want to intercept click events:
    - `trackable`.* Clicks are tracked for all DOM elements with a `trackable` attribute, in the scope of a <track:clicks> element.
    - `data-click-type` (optional) - Specify a custom click type for a trackable DOM element. If `data-click-type` is not specified, the click type defaults to the `type` argument specified in the `<track:clicks>` tag.

- `data-click-url` (optional) – Specify a custom URL to track for a trackable DOM element.

- **Deleting annotations is tracked.** Annotation deletions are now tracked (they generate signals).
- **Restrict tk.session cookie to HTTPS channels.** You can now configure whether or not the `tk.session` cookie should be restricted to HTTPS channels over "SSL" (actually TLS). By default, the cookie is sent over all channels (the prior behavior).
  To restrict the cookie to HTTPS channels, include this entry in `conf/activity/activity.conf`:

```
secure-cookie: true
```

- **Set the hover timeout of a widget popover.** You can set the hover timeout of the widget popover. This is the amount of time in milliseconds that the popover is displayed *after the user leaves the popover trigger*. The optional parameter is `hover-timeout`. The default is 1000 (1 second). Here is an example that sets the hover timeout to 1/10th of a second:

```
<widget:popover mode="hover" hover-timeout="100"> <popover-trigger><button styling="button">Hover with a short timeout></popover-trigger> <popover-content> <h1>Hover</h1> <p>The hover timeout of the button you are hovering over is set to 1/10th of a second.</p> </popover-content> </widget:popover>
```

- **Customize icons for facet lists.** You can customize the icons used to expand and collapse a facet list.
  a. In the *project-name*`/assets/styles.css` file, define CSS class selectors for the collapsed (the first of these) and expanded (the second) icons, for example:

```
.custom-collapse-facet .collapsed a.facet-toggle:after { content: '\e817'; * font-family: "twigkit-basic"; } .custom-collapse-facet a.facet-toggle:after { content: '\e815'; font-family: "twigkit-basic"; }
```

  b. Reference the CSS classes in the `search:facet` tag, for example:

```
<search:facet-list response="response" styling="facet-list facet-list-step facet-list-wrappedheader facet-list-wrappedheader-noborder"> <search:facet class="custom-collapse-facet" facet-name="test1" title="First Pass Files" collapsible="true" show="1" show-more="10"> </search:facet> </search:facet-list>
```

- **Press Enter to save a search query.** After entering a name for a saved search, a user can now press Enter to save the search query.
- **Number formatting for widget:slider.** The `widget:slider` element now supports number formatting. Use a `number-format` parameter to specify a format using the Java

DecimalFormat *(https://docs.oracle.com/javase/7/docs/api/java/text/DecimalFormat.html)*.
For example, for a Cost facet:

```
<widget:slider response="response" facet-name="Cost_i" number-
format="$#,###.00"></widget:slider>
```

- **Performance improvement for search-enabled facets**. Performance has been improved for search-enabled facets, that is, `search:facet` elements with `search-enabled` set to `true`.
- **Choose fonts to include in web apps**. You can choose among three Less stylesheet language files for including fonts in web apps.
  - `phoenix.all.less` (available prior to 4.2.0) - This stylesheet uses custom fonts fetched from a content delivery network by the browser. If a web app is hosted on a secure site with no Internet access, this stylesheet can have font loading errors that will manifest as "using different fonts" and as an eternally loading spinner in the browser somewhere.
  - `phoenix.all.local-fonts.less` (new in 4.2.0) - This stylesheet includes custom fonts that are hosted in the Appkit web app. However, you cannot use this stylesheet when a web app is accessed via the Fusion Proxy and Web Apps services.
  - `phoenix.all.no-fonts.less` (new in 4.2.0) - This stylesheet excludes all custom fonts.

  Suggested usage:
  - Normal client, not using the Fusion Web Apps and Proxy services - `phoenix.all.local-fonts.less`
  - Normal client, using the Fusion Web Apps and Proxy services - `phoenix.all.less`
  - Secure client.* `phoenix.all.no-fonts.less`

  To change the Appkit Less file in an app:
  Find the reference to the stylesheet in `wro.xml` or `twigkit.less`. After the `appkit-ui` Node module is installed, the Less stylesheets are found in `node_modules/appkit-ui/dist/less/`.
- **jQuery 3.3.1 support**. Appkit upgraded its use of jQuery to version 3.3.1
- **Performance improvement for loading comments**. Loading times for comments in app UIs have been reduced significantly.
- **Configuration property passthrough for Elastic X-Pack**. For the Elastic 6 platform, any configuration properties that start with `xpack` are passed through to the Elastic X-Pack client.
- **Modify path of app URL**. To make your app available at a different URL than `http://`***`hostname`***`:*`*`port`*`*/` (for example, `http://localhost:8080/`), use the `-Dtwigkit.http.context` command-line parameter when starting the app to specify a path component to append to the URL. This can make using a reverse proxy easier.
  For example, `-Dtwigkit.http.context=search` would start the app at `http://`***`hostname`***`:*`*`port`*`*/search` (for example, `http://localhost:8080/search`)
  Each app must use a different HTTP port.
- **License restriction in Fusion**. The Appkit Fusion platform is now license-restricted. Using the Fusion platform now requires this license property in the Fusion `license.properties` file:

```
property_social: *
```

- **Sort Solr facets by count**. Previously, in the `SolrHierarchicalFacetService` and `GreedySolrHierarchicalFacetService` services, you could only sort by `name`. Now, you can also sort by `count`.

- **Where to enable tracking**. The determination of whether tracking should be enabled is now made in `ActivityModule` instead of in individual trackers.
  `Tracking is either enabled (the default) or disabled for all trackers.`
  Upgrade information: To upgrade a project in which tracking is disabled to the 4.2.0 version of Appkit, migrate the `enable: false` configuration parameter from `activity/tracking/tracker.conf` to `activity/tracking.conf`.
- **Redirection to logout page when in HTML5 mode**. The `security:user-actions` tag has been improved to redirect to the logout page when the application is in HTML5 mode.
- **Specify hidden fields in search:fields**. In addition to specifying user selectable fields, the `search:fields` element now lets you use the new attribute `hidden-fields` to specify a set of fields that will always be requested, whether or not the user selects them.

Fixed issues

The following issues are fixed in this release:

- When CSRF security is enabled, the backend for Appkit Lightning social/collaboration widgets does't pass the request token to the frontend.
- Search query saved previously in the `search:box` component persists in the saved-search pane and must be cleared manually.
- The `widget:date-range` component lets a user select a To date that precedes a From date.
- Custom colors defined in `@highchart-colors` do not work.
- Clearing all breadcrumbs does not clear the breadcrumb for the search query term.
- Lucene index delete queue is not flushed after document deletion.
- The `chart:histogram` element does not display a histogram if there is only a single facet value.
- The indicator of the sorting order for search results is not set when a new sort is performed.
- For the `search:box` and `advanced:button` elements, pagination is not reset when a new search is performed.
- The `widget:date-range` component uses a previously selected To date when reselecting dates.
- When using both tabs and saved searches, sometimes selecting a saved search had no effect.
- When doing partial updates using an Appkit Solr platform, the `id` field is modified incorrectly.
- The `suggester` parameter in the `SolrSuggestionsRetriever` service is ignored, which prevents specification of multiple suggester dictionaries.
- An OAuth filter conflicts with a Spring Security filter if the filters have the same name.
- Logging about floating licenses is not correct.
- It is impossible to use arrows to select typeahead results when a `search:result-list` element is nested inside an `instant:result-list` element.
- A dependency conflict exists between the Elastic 6 adaptor and the Media module. It is not possible to use them together.
- Track Clicks cannot calculate the offset if RPP (results per page) is not included in the query.
- Hierarchical facet filters are not being fully cleared.
- Attempting to connect to a Fusion data source using a query profile returns a 404 error if the Fusion collection has more than one shard.
- The draw function in Google Maps creates the longitude filter incorrectly.
- When using the `facet:dropdown` element and making more than one facet selection, incorrect facets are selected, highlighted, indicated in breadcrumbs, and used in

queries.

- Calling `store` on the Lucene platform does not add new documents.
- New bounds are not used for a Google Map when a user clicks a filter.
- Documentation for the `collaboration:comment-list` tag mentions bookmarks instead of comments.
- When no LDAP groups are found for a user, an invalid LDAP query cause an exception to be thrown.
- Clicking outside a list of selectable fields displayed by a `search:fields` element does not hide the list of fields.

Date Format processor

The Date Format processor `DateFormatProcessor` lets a user format the display value of a date field.

How to enable `DateFormatProcessor`:

1. Add the file `data-format.conf` under `conf/processors/response`. In the file add:

```
name: twigkit.search.processors.response.DateFormatProcessor
format: date-format
```

   For example:

```
name: twigkit.search.processors.response.DateFormatProcessor
format: dd-MM-yyyy
```

   The default format is `yyyy-MM-dd`.

2. Add the processor to the relevant platform configuration. For example, in `conf/platforms/fusion/data.conf` add:

```
workflow: processors.response.date-format
```

   When this platform is used to request search results, the Date Format processor is used to format dates in the response.

Fusion Analytics Catalog platform

A Fusion Analytics Catalog platform has been added to query against the Fusion Analytics Catalog collection. This platform translates Appkit queries into SQL statements which are executed against a Fusion query pipeline that includes an *Analytics Catalog Stage*. The Fusion Analytics Catalog platform uses the *Fusion Catalog API*.

The Analytics Catalog platform expects standard Fusion platform configuration parameters (for example, `host` and `collection`).

The current implementation only supports fields and facets parameters in the <search:query> tag.

Get started with the Fusion Analytics Catalog platform

1. Start Fusion: `bin/fusion start`
2. Start the Fusion SQL service: `bin/sql start`
3. Add a new platform configuration file, for example `conf/platforms/fusion/analytics.conf`. Include these properties:

```
name: twigkit.fusion.analytics.FusionAnalytics
resultIDField: ID-field
userName: Fusion-user
password: password
host: host-name
collection: Fusion-collection
```

   For example:

```
name: twigkit.fusion.analytics.FusionAnalytics
resultIDField: id
userName: admin
password: password123
host: search.my.org
collection: analytics
```

4. Edit the configuration file `conf/services/api/fusion.conf`. Specify the catalog to use with the platform. The default catalog is `fusion`:

```
catalog-project: fusion-catalog
```

5. Use the platform in HTML pages. For example:

```
<search:platform var="platform" conf="platforms.fusion.analytics">
</search:platform> <search:query var="query" parameters="*"
fields="id,pdf_url_s,summary_t,published_dt pubdate,categories_s"
facets="categories_s,pdf_url_s"></search:query> <search:response var="response"
platform="platform" query="query"> <widget:spinner></widget:spinner>
</search:response>
```

Query Stage SDK

In addition to Fusion's *built-in query stages*, Lucidworks provides the Query Stage SDK for developing your own custom query stages with Java. All of the resources are provided in a public GitHub repository *(https://github.com/lucidworks/query-stage-sdk)*.

Getting started

Clone the repository to get started:

```
git clone https://github.com/lucidworks/query-stage-sdk
```

See Gradle quickstart *(https://docs.gradle.org/current/userguide/tutorial_java_projects.html)* documentation for more information on Java Projects.

The Query Stage SDK requires the *Fusion Search DSL*.

Query Stage SDK library

The Java SDK library contains the classes and interfaces for building new query plugin stages. You'll find it in the `query-stage-plugin-sdk` directory *(https://github.com/lucidworks/query-stage-sdk/tree/master/query-stage-plugin-sdk)*.

Example plugin stage

The example query stage plugin contains an example of how a plugin can be implemented as well as information on how to build and deploy the plugin. You'll find it in the `examples/sample-plugin-stage` directory *(https://github.com/lucidworks/query-stage-sdk/tree/master/examples/sample-plugin-stage)*.

Java SDK Library

The Java SDK library contains the classes and interfaces for building new query plugin stages. You'll find it in the `query-stage-plugin-sdk` directory *(https://github.com/lucidworks/query-stage-sdk/tree/master/query-stage-plugin-sdk)*.

Plugins

A plugin is a `.zip` file that contains one or more query stage implementations. The file contains `.jar` files for stage definitions and additional dependencies. It also contains a manifest file that holds the metadata Fusion uses to run the plugin.

For build and deployment instructions, see *Build and Deploy A Query Stage Plugin*.

Alternatively, it can be assembled by any other build tool as long as the file and directory structure is correct.

Query Stage

The *`com.lucidworks.querying.api.QueryStage`* class is the basic contract for defining a plugin query stage.

A plugin stage class must:

- Implement the *`com.lucidworks.querying.api.QueryStage`* interface.
- Be annotated with the *`com.lucidworks.querying.api.Stage`* annotation.

For convenience, the plugin stage can extend *`com.lucidworks.querying.api.QueryStageBase`*, which already contains initialization logic and some helpful methods.

Example of a plugin query stage implementation:

```
@Stage(type = "simple", configClass = SimpleStageConfig.class) public class SimpleStage extends QueryStageBase<SimpleStageConfig> { private static final Logger logger = LoggerFactory.getLogger(SimpleStage.class); @Override public DslQueryRequestResponse process(DslQueryRequestResponse query, Context context) { queryRequestResponse.getDslQueryRequest().getParams().put(config.queryParam(), config.value()); return queryRequestResponse; } }
```

Lifecycle

1. Fusion creates a `QueryStage` instance and initializes it.
2. Fusion begins calling the `process` method (see the example above) for each query passing through the query pipeline.

One stage instance can be used by Fusion for processing multiple queries and the `process` method can be called from multiple concurrently-running threads. Additionally, Fusion can initialize and maintain multiple stage instances with the same configuration in separate query service nodes. Therefore, it is important to make sure the plugin stage implementation is thread-safe and processing logic is stateless.

The plugin stage may throw an exception while processing a query. This does *not* cause any side effects; the query will simply not be processed anymore. The whole query pipeline will still be in use and Fusion will continue to call the `process` method for other queries. Any information about thrown exceptions can be found in the logs *(https://doc.lucidworks.com/fusion/5.4/130/the-dev-ops-center#log-viewer)*.

Initialization

After creation, each stage object will be initialized using the `init(T config, Fusion fusion)` method. This allows the stage to create any needed internal structure and validate the stage configuration.

Note that initialization occurs immediately after the stage configuration has been saved in Fusion. After this the stage instance can be maintained and re-used by Fusion for

extensive periods of time even if no queries pass through the stage. You should be mindful of this fact when making decisions on resource allocations.

Query-Response processing

After the stage has been initialized, Fusion will start sending queries to the stage for processing by invoking `process(DslQueryRequestResponse query, Context context)` on every incoming query.

Note that multiple threads can call the `process` method concurrently, therefore its implementation must be thread-safe.

Logging

Query Stage SDK is using the slf4j logging API *(http://www.slf4j.org/)*. The following example demonstrates how logging can be implemented inside a plugin stage:

```
public class MyStage extends QueryStageBase<MyStageConfig> { private static final Logger logger = LoggerFactory.getLogger(MyStage.class); @Override public DslQueryRequestResponse process(DslQueryRequestResponse query, Context context) { // ...... logger.info("Processing query '{}'", query); // ...... } }
```

Metrics

Fusion is using Prometheus and Grafana *(https://doc.lucidworks.com/fusion/5.4/131/prometheus-and-grafana-in-fusion)* for enhanced metrics collection and querying. Query Stage plugins can add their own custom metrics to the list of default metrics already generated by the Fusion querying service.

To be able to publish custom metrics the Prometheus client library must be added to the plugin's dependencies:

```
provided 'io.prometheus:simpleclient_dropwizard:0.7.0'
```

After that, the Prometheus client can be used to record custom metrics. More information on the Prometheus Java client API can be found in the Prometheus documentation *(https://github.com/prometheus/client_java#instrumenting)*.

The following code demonstrates how to capture external request time metrics using the Prometheus client API.

```
public class SampleStage { private static final Histogram EXTERNAL_REQUEST_TIME = Histogram.build() .help("Time to execute external query request.") .name("external_query_request_time") .labelNames("request_url") .register(); @Override public DslQueryRequestResponse process(DslQueryRequestResponse query, Context context) { // ...... Histogram.Timer externalQueryTimer = EXTERNAL_REQUEST_TIME .labels(requestUrl) .startTimer(); try { // perform external request... } finally { externalQueryTimer.observeDuration(); } // ...... } }
```

Query stage configuration

The *`QueryStageConfig`* defines configuration options specific to a particular query stage instance. These options will be available to the end user via the Fusion UI and API. The

plugin config class must extend `com.lucidworks.querying.config.QueryStageConfig` and be annotated with `@RootSchema`.

By adding `@Property` and type annotations to your stage configuration interface methods, you can define metadata and type constraints for your plugin configuration fields. This is very similar to Fusion's connector configuration schema. For more detailed information on the configuration and schema capabilities, see Java Connector Development *(https://doc.lucidworks.com/how-to/824/java-connector-development)*.

Here is an example of a simple stage configuration schema definition:

```
@RootSchema( title = "Simple", description = "Simple Query Stage" ) public interface SimpleStageConfig extends QueryStageConfig { @Property( title = "Time allowed", description = "The amount of time allowed for a search to complete.", required = true ) @NumberSchema() long timeAllowed(); }
```

Exposed Fusion APIs

SDK-based plugins are capable of communicating with other Fusion components via the *Fusion* object. This object is passed to the stage during the initialization phase.

RestCall

The *RestCall* API provides access to the Fusion REST API *(https://doc.lucidworks.com/fusion/5.3/8823/rest-api-quick-reference-guide)*.

Blobs

The *Blobs* API is a specialized API for interaction with the Fusion Blob Store API *(https://doc.lucidworks.com/fusion-server/5.3/reference/api/blob-store-api.html)*

Data structures

DslQueryRequestResponse

DslQueryRequestResponse is a representation of a query request and response. In Fusion, both the query request and response are treated and tracked as one entity. This is the class that is passed through to the `process` method and may be manipulated to inspect/update the request and/or response.

DslQueryRequest

DslQueryRequest is a representation of a query request. You can access this via the `QueryRequestAndResponse` class. It contains information about the query parameters, the headers, as well as the HTTP method called.

DslQueryResponse

DslQueryResponse is a representation of a query response. You can access this via the `QueryRequestAndResponse` class. It contains information about the documents, the facets, and any highlighted terms.

Query Stage SDK Javadocs

See the *Javadocs* for complete details about all packages and classes.

Example Query Stage Plugin

The example query stage plugin contains an example of how a plugin can be implemented as well as information on how to build and deploy the plugin. You'll find it in the `examples/sample-plugin-stage` directory *(https://github.com/lucidworks/query-stage-sdk/tree/master/examples/sample-plugin-stage)*.

The sample plugin stage is an example project with two simple querying stages:

- Update request - Add a new query parameter with a specific value to each query request.
- Update response - Add a new field with a specific value to each query response.

This is also a demonstration of a basic `gradle` project that assembles a Fusion query stage plugin `.zip`.

1. Build the plugin stage

From the main folder, call:

```
./gradlew -p examples/sample-plugin-stage clean assemblePlugin
```

Substitute the path to your project.

This creates a plugin `.zip` file (with required manifest file) inside the `build/libs` folder.

2. Deploy the stage

You can choose from several different ways to deploy the sample plugin stage:

- *Upload it to the blob store*
- *Use gradle*
- *Use the REST API*

After successful deployment, new stages should be visible in the `Stages` list in the Fusion Query Pipelines UI.

Upload it to the blob store

1. Navigate to **System** > **Blobs**.
2. Click **Add**.
3. Select **Query Stage Plugin**.
4. Click **Browse…** and select your plugin file.
5. Click **Upload**

Deploy it using gradle

```
./gradlew -p examples/sample-plugin-stage deploy -PfusionUser=[user] -PfusionPassword=[password]
```

Deploy it with the REST API

```
curl -u [user]:[password] -X PUT -H "Content-Type:application/zip" --data-binary
@sample-plugin-stage-0.0.1.zip https://<fusion-host>/api/query-stage-plugins
```

Jupyter Support in Fusion

Jupyter *(https://jupyter.org/)* is an open-source web application that allows you to create and share documents that contain live code, equations, visualizations, and narrative text.

Fusion 5 provides a Jupyter service that can be run from the Fusion Helm chart. See *How to enable Jupyter* and *How to use Jupyter with Fusion SQL*.

This feature is not available in Managed Fusion environments.

Why Jupyter

Jupyter is evolving, and it has a number of kernels that support different programming languages. With Jupyter, you can run Spark in Scala or Python and run Fusion SQL at the same time in Python. This versatility gives you options to debug and try out various Fusion features. By integrating BeakerX *(http://beakerx.com/)*, we get access to a wide variety of kernels and visualization features. And by configuring Jupyter, we can hide it behind the Fusion gateway (proxy) service and avoiding exposing the Jupyter IP externally.

What this service is for

- Run/Debug Spark code in Scala/Python (Replacement for spark-shell)
- Run SQL queries via Fusion SQL
- Debug Scala and SQL transforms in PBL jobs
- Everything else for which Jupyter is designed

What this service is not for

This service is not for running Spark jobs in a Kubernetes cluster.

The Jupyter pod does not have access to create or delete pods in a Kubernetes cluster, and therefore you cannot run jobs from Jupyter notebook in a Kubernetes cluster. However, Spark local mode can be used with a higher driver memory if needed.

We recommend sampling data for debugging Jupyter and then running the actual job using the *Fusion jobs UI*.

Configuration tips

- The Jupyter pod IP is `ClusterIP` and not exposed externally. We recommend accessing it via the Fusion proxy and not exposing the pod IP via `LoadBalancer`.
- Jupyter can also be accessed via port-forwarding and available via `localhost:8888/jupyter`
- Idle notebooks and kernels are shutdown after 60 mins of inactivity. Note that any additional libraries installed via `pip` will need to be re-installed when the pod is recycled.
- This image is baked with Python 3.6+.

JupyterHub

Customers who run JupyterHub can use `lucidworks/fusion-jupyter` *(https://github.com/lucidworks/fusion-jupyter)* to launch notebooks with the Fusion Jupyter image.

Release Notes

Release notes provide detailed descriptions of the relevant changes included with each Fusion release, including new features, improvements, bug fixes, and more.

Upgrades

Keeping your Fusion deployment upgraded to the latest version enables you to take advantage of the latest features, improvements, bug fixes, and addressed security issues.

See *Fusion 5 Upgrades* for more information on the upgrade process.

Deprecations and Removals

Every Fusion release adds or improves Fusion features and functionality. Occasionally, Lucidworks "deprecates" features, and they are no longer maintained, updated, or supported. Deprecated features should be planned for eventual removal. In many cases, new features are introduced which replace or improve the functionality of the deprecated feature.

See *Deprecations and Removals* for a list of active deprecations and removals.

How long do I have before the feature is removed?
Lucidworks gives a minimum of one minor release cycle before removing a deprecated feature. For example:

- A feature is deprecated in Fusion 5.2.0, so it will not be removed before Fusion 5.3.0
- A feature is deprecated in Fusion 5.2.1, so it will not be removed before Fusion 5.3.0

	Deprecations are typically made in major and minor releases only. Maintenance release deprecations are less common.

Fusion 5.12.0

Release date: March 28, 2024

Component versions:

Component	Version
Solr	fusion-solr 5.12.0 *(based on Solr 9.4.0)*
ZooKeeper	3.9.0
Spark	3.4.1
Kubernetes	GKE, AKS, EKS 1.28, 1.27, and 1.26 Rancher (RKE) and OpenShift 4 compatible with Kubernetes 1.28, 1.27, and 1.26 OpenStack and customized Kubernetes installs not supported. See *Kubernetes support* for end of support dates.
Ingress Controllers	Nginx, Ambassador (Envoy), GKE Ingress Controller Istio not supported.

> Looking to upgrade?
> Check out the *Fusion 5 Upgrades topic* for details.

New Features

Fusion

JSON viewer for Fusion objects

A new JSON viewer has been introduced in Fusion, providing a user-friendly interface to inspect the JSON configuration of key objects like index pipelines, query pipelines, jobs, and datasources. This viewer allows users to:

- **Visualize configuration:** View the complete JSON representation of the selected object's configuration in a clear and well-formatted manner. For example, a basic datasource configuration resembles the following:


```
{ "id": "mens-products", "created": "2024-02-12T15:08:29.332Z", "modified": "2024-02-12T15:08:29.332Z", "connector": "lucid.fileupload", "type": "fileupload", "pipeline": "Documentation", "parserId": "_system", "properties": { "collection": "Documentation", "fileId": "mens-products.csv", "mediaType": "application/octet-stream" } }
```


- **Facilitate operations:** Easily copy the displayed JSON for use in essential activities such as API calls.
- **Download for reference:** Download the JSON configuration as a file for backup or collaboration.

To learn how to use the JSON viewer, see the following demonstration:

Asynchronous parsing service

An asynchronous parsing service for connectors, Apache Tika Container, has been added. This stage is now required when using asynchronous parsing for connectors. To use asynchronous parsing for connectors, be sure that the **Async Parsing** is checked in the datasource and the Apache Tika Container parser stage is enabled in the index pipeline.

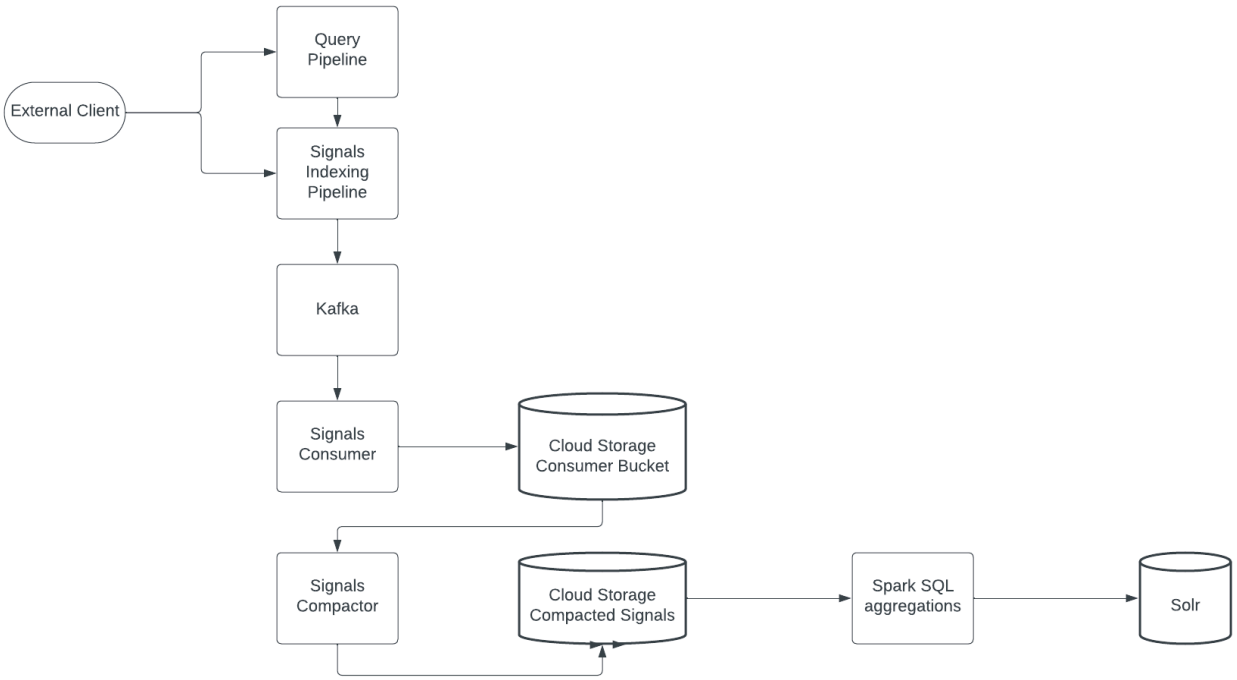
	Other parsers, such as HTML and JSON, are now supported by the asynchronous parsing service. By enabling asynchronous parsing, the parser configuration linked to your datasource is used.

To learn how to start using the new asynchornous parsing service, see the following demonstration:

Managed Fusion

Cloud signal storage

Fusion now offers the flexibility to store signals data in Google Cloud Storage or Amazon S3, providing an alternative to the default Solr storage. Cloud signal storage minimizes the storage burden on your Solr cluster, potentially leading to improved performance and resource optimization.



All signal types, including custom types, are fully supported for cloud storage. Additionally, customization of the default signal schema remains available. Signals data files are automatically compacted over time into larger files, resulting in significant storage space savings and simplified file management.

Cloud signal storage is enabled during initial deployment. For more information, see *Cloud signal storage*.

Improvements

- Introduced significant improvements to job status reporting within Fusion. A dedicated job function microservice, `job-config`, has been added to control job execution and store job history. This microservice requires no configuration from the client. In addition, to optimize communication efficiency, synchronous calls have been replaced with asynchronous communication through Kafka. As a result of these changes, job reports are now far more accurate and reliable than before.
  These changes do not impact Fusion API usage or user workflows.
- Introduced a configurable timeout for datasource jobs, in case the connection to the backend job is lost. Previously, the job was stopped after a fixed duration of 100 seconds.
- Previously query pipeline pods could restart due to scaling or other reasons, leading to 503 errors in the API gateway. This release introduces a configurable retry setting for the api-gateway service that will automatically attempt to reconnect to the query pipeline pods. This addresses the previously observed 503 errors under sustained query load.
- Pre-packaged configurations for commonly used dimensionalities (64, 384, 512, and 768) in vector search operations within Solr are added to Fusion. These configurations leverage a naming convention that incorporates dimensionality into dynamic field names, simplifying setup and management for users. For example:

```
<fieldType class="solr.DenseVectorField" codecFormat="Lucene90HnswVectorsFormat"
hnswBeamWidth="40" hnswMaxConnections="10" name="knn_512_vector"
similarityFunction="cosine" vectorDimension="512" /> <dynamicField
docValues="false" indexed="true" multiValued="false" name="*_512_v"
required="false" stored="false" type="knn_512_vector" />
```

- Introduced automated management of Kafka topic partitions for indexing subscriptions within Fusion. Fusion now intelligently determines the optimal number of partitions, creates the topics with appropriate configurations, and initiates the subscription process.

Bug Fixes

- Fixed a bug that prevented remote connector datasources from indexing data after encountering a "No Plugin Activity" error. Previously, affected datasources required recreation with a different name despite maintaining identical specifications.
- Fixed a bug that caused a Solr syntax error when attempting to delete an aclDocument using the SDK method `newDeleteGraphAccessControlItem` if the ID of the document contained special characters.
- Fixed a bug that sometimes resulted in a `ConcurrentModificationException` error during high-load testing of query pipelines utilizing Solr Subquery and Merge Async Results stages.
- Addressed a race condition impacting dynamic pricing functionality. Previously, concurrent user requests for the same external file could lead to the second request encountering an error. The fix ensures proper handling of concurrent requests, eliminating the potential for this error.
- Fixed a bug that caused a `ConcurrentModificationException` error during the `.RawPull` replication process within Solr.
- Fixed a bug in the UI login process where attempting to log in with an incorrect password after an expired session resulted in no error message being displayed on the UI.

Deprecations

V1 Fusion connectors

Starting in Fusion 5.12.0, all V1 connectors are deprecated. This means they are no longer being actively developed and will be removed in the next LTS release, Fusion 5.13.0. Please note:

- **Some V1 connectors already have a direct replacement available.** You can find the replacement connector on the *Fusion Connectors Deprecations and Removals page*.
- **For all other V1 connectors, a replacement is still under development.** We will update the documentation when the replacement is available.

If you are using a V1 connector, you must migrate to the replacement connector or a supported alternative before upgrading to Fusion 5.13.0. We recommend migrating to the replacement connector as soon as possible to avoid any disruption to your workflows.

Banana dashboards

Banana, the open-source browser visualization tool that previously powered Fusion's Banana Dashboards, is deprecated for Fusion. It is scheduled for removal in Fusion 5.13.0. While Banana will no longer be included with Fusion, users who require continued use can leverage a separate Docker deployment option. This alternative deployment method will be available in Fusion 5.13.0.

If you would like to explore the open-source project, see Banana on Github *(https://github.com/lucidworks/banana)*.

Other deprecations

- App Insights and the insights microservice are deprecated and will be removed in a future release.
- The Parsers CRUD API from the indexing service is deprecated and will be removed in a future release. Use `/async-parsing/` instead.
- The Train a Smart Answers cold start model job is deprecated and will be removed in a future release. Use one of the pre-trained models or the supervised training job instead.
- The Data Augmentation job is deprecated and will be removed in a future release.

Fusion 5.11.0

Release date: January 3, 2024

Component versions:

Component	Version
Solr	fusion-solr 5.11.0 *(based on Solr 9.4.0)*
ZooKeeper	3.9.0
Spark	3.4.1
Kubernetes	GKE, AKS, EKS 1.27 Rancher (RKE) and OpenShift 4 compatible with Kubernetes 1.27 OpenStack and customized Kubernetes installs not supported. See *Kubernetes support* for end of support dates.
Ingress Controllers	Nginx, Ambassador (Envoy), GKE Ingress Controller Istio not supported.

More information about support dates can be found at *Lucidworks Fusion Product Lifecycle*.

> Looking to upgrade?
> Check out the *Fusion 5 Upgrades topic* for details.

New Features

Fusion

- The *App Manager API* provides programmatic access to information about your Fusion license, entitlements, and usage data. This data is also used to generate usage statistics in the *Consumption Dashboard*.
  For example, send a request to the `/apps-manager/entitlements/current` endpoint to retrieve general entitlements usage data:
  Input

```
curl -u USERNAME:PASSWORD https://FUSION_HOST:FUSION_PORT/apps-manager/entitlements/current
```

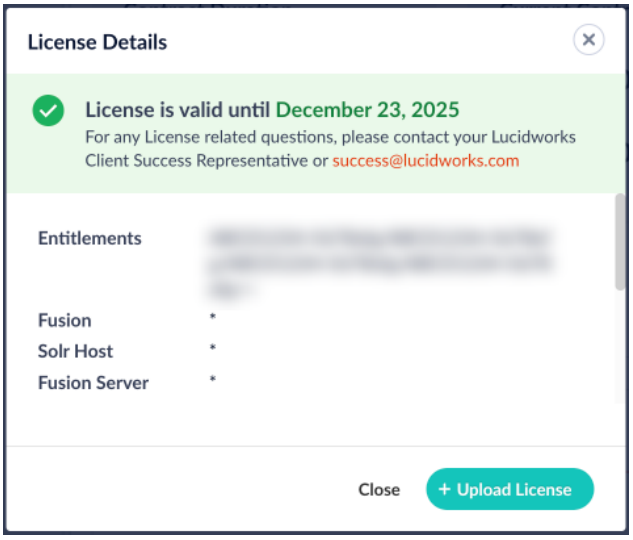
  Output

```
{ "contractDuration":{ "years":3, "months":0, "days":0 }, "info":{ "start":"2022-12-23", "end":"2023-12-23", "requests":10000000, "records":5000000 }, "usage":{ "requests":2863, "requestsPercentage":0, "records":565672, "recordsPercentage":11 } }
```

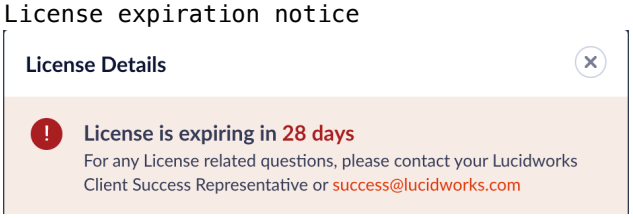
Improvements

Fusion

- Solr has been upgraded to 9.4. For details about changes in Solr 9.4, see Apache Solr Release Notes *(https://solr.apache.org/guide/solr/latest/upgrade-notes/major-changes-in-solr-9.html#solr-9-4)*.
- Fusion now supports Kubernetes 1.27. This applies to GKE, AKS, and EKS. It also applies to Rancher (RKE) and OpenShift 4 versions that are compatible with Kubernetes 1.27. Refer to Kubernetes documentation at Kubernetes v1.27 *(https://kubernetes.io/blog/2023/04/11/kubernetes-v1-27-release/)* for more information.
- Fusion safeguards against indexing overload with an ingestion flow control system. This system prevents timeouts by:
  - Logging all Kafka topics used for indexing to aid in diagnosing performance issues.
  - Monitoring the rate of data production and consumption for each Kafka topic across all indexing jobs.
  - Automatically throttling producers to ensure consumers can keep up.
- The *Consumption Dashboard* has been updated to restore three charts.
- The Fusion UI is updated with improved messaging around Fusion license renewal and expiration dates.
  License details



  License expiration notice



- The Fusion API Gateway now includes an optional Kubernetes health check. To enable this check, edit your `values.yaml` file, and add `-Dspring.cloud.kubernetes.loadbalancer.mode=POD` to the API Gateway's `javaToolOptions`. For example:

```
fusion: api-gateway: javaToolOptions: "-Dspring.cloud.kubernetes.loadbalancer.mode=POD"
```

  When enabled, if the health check fails during pod discovery, Kubernetes will restart the pod.

- The fusion-solr Helm chart now accepts user-defined system properties that are set in `values.yaml`. In the `values.yaml` file, populate the `solrEnvExtra` array with entries for each desired variable. Each variable requires a name and either a direct value or a value source (`valueFrom`). These variables will be included in the final configuration in the order you provide them, following any variables dynamically generated by the Helm chart templates. For more information on Kubernetes environment variables, see Define Dependent Environment Variables *(https://kubernetes.io/docs/tasks/inject-data-application/define-interdependent-environment-variables/)*.
- The Fusion migration script is updated to align with changes from the Solr upgrade. The migration script:
  - Removes the unused configuration, `<circuitBreaker>`, from `solrconfig.xml`. Solr no longer supports this configuration.
  - Detects if `StatelessScriptUpdateProcessorFactory` is found in `solrconfig.xml`. The migration script will update the configuration, if possible, and provide guidance on actions you must take.

Fusion connectors

- The default number of fetch threads for V2 connectors is now 5. Previously, the default was 20. This default setting was reduced because some Fusion environments experienced failed datasource crawls as a result of too many requests in a short period of time.

Bug Fixes

- The *Configurations* and *ZooKeeper Import/Export* APIs are now accessible only to authorized users from within your Fusion cluster or through Kubernetes port forwarding.
- The Fusion API realm can now authenticate when an API key is passed via an `apiKey` query parameter. Previously, this only worked via the `x-api-key` header.
- The ZooKeeper Docker image is updated to use Logback *(https://logback.qos.ch/)* for logging instead of log4j. This change fixes a bug that forced all Zookeeper log messages to use the DEBUG level.
- Fixed a bug that sometimes resulted in an HTTP 431 error when using the Fusion admin UI.
- Fixed a bug that sometimes resulted in an HTTP 414 error when using the Rules UI.

Deprecations

- Fusion has deprecated the Webapps service. In previous versions of Fusion, you could use this service to deploy an App Studio WAR file into Fusion. As of Fusion 5.11.0, you should instead *deploy App Studio manually*.

Removals

- The previously non-functional Subscriptions UI has been removed. This is related to the removal of the *Subscriptions API* in Fusion 5.6.0.
- MLeap is removed from the ml-model service. MLeap was deprecated in Fusion 5.2.0 and was no longer used by Fusion.

Fusion 5.10.0

Release date: September 28, 2023

Component versions:

Component	Version
Solr	fusion-solr 5.10.0 *(based on Solr 9.3.0)*
ZooKeeper	3.8.1
Spark	3.4.1
Kubernetes	GKE, AKS, EKS 1.25 and 1.26 Rancher (RKE) and OpenShift 4 compatible with Kubernetes 1.25 and 1.26 OpenStack and customized Kubernetes installs not supported. See *Kubernetes support* for end of support dates.
Ingress Controllers	Nginx, Ambassador (Envoy), GKE Ingress Controller Istio not supported.

More information about support dates can be found at *Lucidworks Fusion Product Lifecycle*.

	Looking to upgrade? Check out the *Fusion 5 Upgrades topic* for details.

New Features

Fusion Connectors

- The new Adobe Experience Manager (AEM) V2 connector has been added alongside the Fusion 5.10.0 release. The AEM V2 connector is compatible with Fusion 5.5.1 and later and works with AEM version 6.5. For more information, including configuration details, see *AEM V2 Connector Configuration Reference*.

Improvements

Fusion

- Solr has been upgraded to 9.3.
- The consumption dashboard now features a summary section that displays your total entitlements, requests, and records. You can also view your usage over your contract year.
- New user roles for Fusion have been created to enhance security. Each of these roles should be combined with the developer role in order to work.
  - The spark-developer role can create, edit, and delete Spark jobs.
  - The script-developer role can create, modify, and delete JavaScript code in the querying and indexing pipelines.
  - The stage-plugin-developer role can create, edit, and delete custom index and query pipeline stages.
- The GraalVM JavaScript stage has been added to the index and query pipelines. It is available in Nashorn compatibility mode, but full backwards compatibility is not available. The Nashorn JavaScript library is still the default option. To use the GraalVM library, select **GraalVM** from the Javascript engine dropdown in the Fusion UI.

This is an experimental feature and should not be used in production. Remote calls to other services are not available.

- The *Graph Security Trimming stage* has new default values for `aclSolrCollection`, `joinField`, and `joinMethod`. Previously deprecated fields `crossCollection` and `dvWithScore` are removed.
- Managed Fusion only: The reverse search saved-query stage now removes any `partition_filter` fields from saved query documents that consist of the empty string.

Bug Fixes

- Fixed a bug where a non-admin user with the proper permissions could not type in the pipeline configuration source code editor.
- Fixed a bug where the Context object used when posting each document in an array to an index pipeline was the same object.
- Fixed a bug where asynchronous parsing was unable to push documents to Kafka when TLS is enabled.

Removals

- The Forked Apache Tika parser stage has been removed. Use *asynchronous Tika parsing* instead.
- The Analytics Catalog query stage has been removed.

Fusion 5.9.3

Release date: February 26, 2024

Component versions:

Component	Version
Solr	fusion-solr 5.9.3 *(based on Solr 9.1.1)*
ZooKeeper	3.7.1
Spark	3.2.2
Kubernetes	GKE, AKS, EKS 1.28 Rancher (RKE) and OpenShift 4 compatible with Kubernetes 1.28 OpenStack and customized Kubernetes installs not supported. See *Kubernetes support* for end of support dates.
Ingress Controllers	Nginx, Ambassador (Envoy), GKE Ingress Controller Istio not supported.

More information about support dates can be found at *Lucidworks Fusion Product Lifecycle*.

	Looking to upgrade? Check out the *Fusion 5 Upgrades topic* for details.

Improvements

- Fusion now supports Kubernetes 1.28. This applies to GKE, AKS, and EKS. It also applies to Rancher (RKE) and OpenShift 4 versions that are compatible with Kubernetes 1.28. Refer to Kubernetes documentation at Kubernetes v1.28 *(https://kubernetes.io/blog/2023/08/15/kubernetes-v1-28-release/)* for more information.

Bug fixes

- Fixed an issue affecting remote connector datasources where they could enter a state of persistent inability to index following a `No Plugin Activity` timeout error.
- Fixed an issue with the async-parser where large files exceeding the configured memory buffer were repeatedly retried for processing, even though they would ultimately fail. This behavior could lead to unnecessary resource consumption and logging noise. Files that exceed the size configuration are no longer retried.
- Fixed an issue with the async-parser where documents containing empty values were erroneously emitted for indexing, even though the indexing process couldn't handle them. This behavior led to indexing failures and unnecessary resource consumption.
- Fixed an issue with the Rules UI where very large rule sets could trigger URI errors, preventing the UI from loading or functioning correctly. This required manual deletion of rules to restore functionality.
- Fixed an issue where previously denied and published rules reappeared as new suggestions after running detection jobs.

Fusion 5.9.2

Release date: December 13, 2023

Component versions:

Component	Version
Solr	fusion-solr 5.9.1 *(based on Solr 9.1.1)*
ZooKeeper	3.7.1
Spark	3.2.2
Kubernetes	GKE, AKS, EKS 1.25, 1.26, and 1.27 Rancher (RKE) and OpenShift 4 compatible with Kubernetes 1.25, 1.26, and 1.27 OpenStack and customized Kubernetes installs not supported. See *Kubernetes support* for end of support dates.
Ingress Controllers	Nginx, Ambassador (Envoy), GKE Ingress Controller Istio not supported.

More information about support dates can be found at *Lucidworks Fusion Product Lifecycle*.

	Looking to upgrade? Check out the *Fusion 5 Upgrades topic* for details.

Improvements

- Fusion now supports Kubernetes 1.27. This applies to GKE, AKS, and EKS. It also applies to Rancher (RKE) and OpenShift 4 versions that are compatible with Kubernetes 1.27. Refer to Kubernetes documentation for version 1.27 *(https://kubernetes.io/blog/2023/03/17/upcoming-changes-in-kubernetes-v1-27/)*.
- Query service replica scaling has been improved.

Fusion Connectors

- Updated the Web V1 (classic) connector library to resolve issues for certain JavaScript-enabled web pages. While Fusion was not vulnerable to the Common Vulnerabilities and Exposures (CVE) 2023-4863, the connector updates allowed Lucidworks to also acknowledge and resolve that publicly-reported CVE.

Bug Fixes

- The connectors-backend service no longer fails to start when deploying to Openshift 4.12.
- Fixed an issue where the Transport Layer Security (TLS) for Fusion Microservices could not be enabled and used.

Fusion 5.9.1

Release date: September 5, 2023

Component versions:

Component	Version
Solr	fusion-solr 5.9.1 *(based on Solr 9.1.1)*
ZooKeeper	3.7.1
Spark	3.2.2
Kubernetes	GKE, AKS, EKS 1.25 and 1.26 Rancher (RKE) and OpenShift 4 compatible with Kubernetes 1.25 and 1.26 OpenStack and customized Kubernetes installs not supported. See *Kubernetes support* for end of support dates.
Ingress Controllers	Nginx, Ambassador (Envoy), GKE Ingress Controller Istio not supported.

More information about support dates can be found at *Lucidworks Fusion Product Lifecycle*.

	Looking to upgrade? Check out the *Fusion 5 Upgrades topic* for details.

Bug Fixes

Fusion Connectors

- Fixed an issue to make gRPC max outbound message size configurable to prevent remote connectors from failing on large field sizes.
- Fixed an issue where a restart was required when the remote JDBC V2 connector timed out and did not restart correctly.
- Fixed an issue where the JDBC V2 connector failed to index data.
- Fixed an issue where the Connectors SDK delete-by-query that contains a NOT-EQUAL phrase did not delete the appropriate documents.
- Fixed an issue where the outbound access control list (ACL) object contained a reference to itself in the `outbound_ss` field by default. The plug-in that creates the ACL should enter that reference explicitly only when required.
- Fixed an issue where the connectors-backend service did not add the following fields by default to the LDAP ACL documents: `_lw_data_source_type_s`, `_lw_batch_id_s`, `_lw_data_source_collection_s`, and `_lw_block_start_time_l`. These fields are required by the LDAP plug-in to keep the list of ACL objects up-to-date.

Fusion 5.9.0

Release date: June 29, 2023

Component versions:

Component	Version
Solr	fusion-solr 5.9.0 *(based on Solr 9.1.1)*
ZooKeeper	3.7.1
Spark	3.2.2
Kubernetes	GKE, AKS, EKS 1.25 and 1.26 Rancher (RKE) and OpenShift 4 compatible with Kubernetes 1.25 and 1.26 OpenStack and customized Kubernetes installs not supported. See *Kubernetes support* for end of support dates.
Ingress Controllers	Nginx, Ambassador (Envoy), GKE Ingress Controller Istio not supported.

More information about support dates can be found at *Lucidworks Fusion Product Lifecycle*.

Looking to upgrade?
Check out the *Fusion 5 Upgrades topic* for details.

Upgrades from Fusion 5.8.0
Fusion 5.8.1 fixed an issue with the Fusion 5.8.0 Helm chart that prevented horizontal pod autoscaling.

If you are upgrading from Fusion 5.8.0 to any later version and use horizontal pod autoscaling, you must complete the procedure detailed in the *Fusion 5.8.1 release notes*.

New Features

- The new *Consumption Dashboard* tracks your request and document usage against the allotted consumption in your Fusion license. To access the dashboard, log into Fusion and go to **Analytics > Consumption**.

Improvements

Fusion

- Fusion now supports Kubernetes 1.25 and 1.26. This applies to GKE, AKS, and EKS. It also applies to Rancher (RKE) and OpenShift 4 versions that are compatible with Kubernetes 1.25 and 1.26. Refer to Kubernetes documentation for versions 1.25 *(https://kubernetes.io/blog/2022/08/04/upcoming-changes-in-kubernetes-1-25/)* and 1.26 *(https://kubernetes.io/blog/2022/11/18/upcoming-changes-in-kubernetes-1-26/)* to decide which version to use for your deployment.
- Added a new Admin button to the Rules Editor screen. Click on it to log out of Fusion.
- Improved Spark Solr's performance to ingest and transport data, as well as setting the most efficient sort for data export.

Fusion Connectors

- Remote V2 connectors now support Tika Asynchronous Parsing. You can *enable Tika Asynchronous Parsing* on *remote V2 connectors* to separate document crawling from document parsing, which is useful for large sets of complex documents.
- Updated the connectors-sdk library version to Java 11.
- Updated Fusion to prevent automatically updating V2 connector plug-ins. Intentional updates allow you to verify related functions and operations.
- Updated Fusion to specify pod resources for connector plug-in type to ensure optimal capability and performance.

Other changes

Basistech Compatibility

- Fusion 5.9.0 and later can reference a new Solr 9.0/9.1 image that comes bundled with Basistech JAR files.

Fusion Connectors

- Fusion 5.9.0 introduces new version requirements for the LDAP ACLs V2 and SharePoint Optimized V2 connectors. The *LDAP ACLs V2 connector* must use version 1.5.0 or later, and the *SharePoint Optimized V2 connector* must use version 1.6.0 or later.

Bug Fixes

Fusion

- Fixed an issue where the parsing endpoint in the indexing service did not parse a DOC file or other Microsoft Word documents. The Apache POI was updated with the `log4j` API.

Fusion Connectors

- Fixed an issue where clearing the `crawldb` for the Access Control List (ACL) datasource also deletes the ACLs.
- Fixed an issue where the Box.com V2 connector did not start successfully when a Fusion branch was deployed.

Fusion 5.8.1

Release date: June 8, 2023

Component versions:

Component	Version
Solr	fusion-solr 5.8.1 *(based on Solr 9.1.1)*
ZooKeeper	3.7.1
Spark	3.2.2
Kubernetes	GKE, AKS, EKS 1.24 Rancher (RKE) and OpenShift 4 compatible with Kubernetes 1.24 OpenStack and customized Kubernetes installs not supported. See *Kubernetes support* for end of support dates.
Ingress Controllers	Nginx, Ambassador (Envoy), GKE Ingress Controller Istio not supported.

More information about support dates can be found at *Lucidworks Fusion Product Lifecycle*.

> Looking to upgrade?
> Check out the *Fusion 5 Upgrades topic* for details.

> Rosette Entity Extractor (REX) or Rosette Base Linguistics (RBL) for the *Basistech language module* or are not compatible with Solr 9 included in this version of Fusion. If you rely on the Basistech language module, do not upgrade until this compatibility issue is resolved.

Bug Fixes

Fusion 5.8.1 fixes a bug in the Fusion 5.8.0 Helm chart that prevented horizontal pod autoscaling from working.

If you are not using horizontal pod autoscaling, you do not need to upgrade to Fusion 5.8.1. This release does not make any other changes to your deployment.

To use horizontal pod autoscaling in Fusion 5.8.1, follow these steps:

1. Add the metrics server to your Fusion 5.8.1 deployment.
2. Ensure the following changes are made your custom values YAML file for horizontal pod autoscaling:
   a. Add service limits to the `resources` object of a service. For example:

```
query-pipeline: resources: limits: cpu: "2" memory: "2Gi" requests: cpu: "500m" memory: "1Gi"
```

> Resource limits vary depending on your deployment. Use values specific to your needs.

Do not use horizontal scaling with Argo.
Horizontal scaling is not available with the Argo workflow controller. Do not apply horizontal autoscaling, or any other type of horizontal scaling, to the Argo service.

Applying horizontal autoscaling to Argo can cause unexpected behavior, such as pods being unnecessarily terminated, jobs failing to launch, or models failing to deploy.

b. Use the new keys and values to support autoscaling. For example, the key `targetAverageUtilization` is now `target`, which requires the keys `type` and `averageUtilization`:

```
autoscaling: enabled: true minReplicas: 1 maxReplicas: 5 metrics: - type:
Resource resource: name: cpu target: type: Utilization averageUtilization: 10
```

To verify that horizontal pod autoscaling is running, use the `k get HorizontalPodAutoscaler` command. The output should resemble the following:

```
NAME                REFERENCE                      TARGETS   MINPODS   MAXPODS
REPLICAS   AGE

f5-query-pipeline   Deployment/f5-query-pipeline   1%/10%    1         5
1          84m
```

Fusion 5.8.0

Release date: March 22, 2023

Component versions:

Component	Version
Solr	fusion-solr 5.8.0 *(based on Solr 9.1.1)*
ZooKeeper	3.7.1
Spark	3.2.2
Kubernetes	GKE, AKS, EKS 1.24 Rancher (RKE) and OpenShift 4 compatible with Kubernetes 1.24 OpenStack and customized Kubernetes installs not supported. See *Kubernetes support* for end of support dates.
Ingress Controllers	Nginx, Ambassador (Envoy), GKE Ingress Controller Istio not supported.

More information about support dates can be found at *Lucidworks Fusion Product Lifecycle*.

Looking to upgrade?
Check out the *Fusion 5 Upgrades topic* for details.

Rosette Entity Extractor (REX) or Rosette Base Linguistics (RBL) for the *Basistech language module* are not compatible with Solr 9 included in this version of Fusion. If you rely on the Basistech language module, do not upgrade until this compatibility issue is resolved.

New Features

- Managed Fusion only: Added a new feature called Dynamic Pricing which improves scalability for custom pricing. This feature lets B2B organizations with large product and pricing inventories sort, facet, boost, and filter on custom prices and entitlements.
- Managed Fusion only: Fusion now supports Reverse Search, which lets you set up monitoring queries that automatically include new documents. Instead of running a query multiple times to see if new documents have been added, this feature matches incoming documents to existing relevant queries, improving content awareness and productivity.
- Tika Asynchronous Parsing improves document crawl speeds and prevents memory and stability issues during connector processes. You can *migrate to asynchronous parsing* to separate document crawling from document parsing, which is useful for large sets of complex documents. For more information, see *Asynchronous Tika Parsing*.

  The Apache Tika and Forked Tika stages are now deprecated. Follow the migration steps to begin using asynchronous parsing.

Improvements

- Improved recoverability for *on-prem connectors* in high network traffic environments.
- Fusion's custom Solr image has been updated to fusion-solr 5.8.0. This upgrade includes the benefits and new features in Solr 9 *(https://solr.apache.org/guide/solr/latest/upgrade-notes/major-changes-in-solr-9.html)*, while also including custom plugins to support Dynamic Pricing, Reverse Search, and autoscaling.

- Developed new authentication methods for the *MongoDB connector*.

Bug Fixes

Fusion

- Fixed a bug where the indexing service failed to load some classes from some JDBC drivers.
- Updated the Helm charts used when deploying Prometheus, Grafana, Loki, and Promtail for monitoring.
- Fixed an error with permissions required for the *Upload Model Parameters To Cloud* job.
- *Graph Security Trimming* stage now works when collections have multiple shards and replicas.
- Fixed a bug where having the same document updated twice in the same job could cause the job to hang.
- Fixed an issue where the *Solr API* was unable to pass through raw requests using the proxy.
- Updated the query pipeline and indexing container base images to use Java 11 so they are more secure.
- Removed UI link to view logs dashboard as its target screen is no longer available.
- Fixed a UI bug where zone display fields could not be manually removed.
- Fusion panel text editors can now scroll as expected in Firefox.

Predictive Merchandiser

- Fixed a bug in *Predictive Merchandiser* where templates having a higher precedence using a specified trigger phrase and facet were not appearing when that phrase was searched with that facet selected.

Deprecations

- *Field Parser Index Stage* is no longer used by Fusion connectors. It is officially deprecated in this release and will be removed entirely in a later release.
- Streaming documents to the `/index` and `/reindex` endpoints of the *Index Pipelines API* is deprecated and will eventually stop working altogether in the continuing switch to asynchronous parsing.
- *Tika Server Parser* is deprecated and will be replaced by Tika Asynchronous Parser.
- *Apache Tika Parser* stage is deprecated and will be removed in a later release.
- The *Forked Apache Tika Parser* stage is deprecated and will be removed completely in a later release.

Known Issues

- When using the JavaScript query stage to query Solr, you have to provide parameters including `rows`. Previously `rows` accepted an integer, but now it must be entered as a string as in `("rows", "1")`.

```
map.put("q", 'okta_uid_s:"{identifier}" OR username_s:"{identifier}" OR email_s:"
{identifier}"' .replace(/{identifier}/ig, userIdentifier)) map.put("rows", "1")
```

Fusion 5.7.1

Release date: January 30, 2023

Component versions:

Component	Version
Solr	fusion-solr 8.11.1 *(based on Solr 8.11.1)*
ZooKeeper	3.7.1
Spark	3.2.2
Kubernetes	GKE, AKS, EKS 1.24 Rancher (RKE) and OpenShift 4 compatible with Kubernetes 1.24 OpenStack and customized Kubernetes installs not supported. See *Kubernetes support* for end of support dates.
Ingress Controllers	Nginx, Ambassador (Envoy), GKE Ingress Controller Istio not supported.

More information about support dates can be found at *Lucidworks Fusion Product Lifecycle*.

	Looking to upgrade? Check out the *Fusion 5 Upgrades topic* for details.

Improvements

- Remote connector jobs now have the ability to recover from temporary network interruptions.

Bug Fixes

- Fixed a bug that caused errors with the Argo workflow due to an issue in the Helm chart.
- Fixed a bug that can prevent the Fusion indexing microservices from running with Transport Layer Security (TLS) enabled. Other microservices, which rely on the indexing microservice, were subsequently prevented from running.
- Fixed a bug with *Spark jobs* that prevented the deletion of the temporary `job-impl.jar` file. This bug resulted in excessive pod disk space usage. Now, the directory containing the file is deleted when the job finishes.
- Fixed a bug with the Smart Answers Supervised Training job that prevented a trained model from deploying.
- Fixed a bug in the Spark 3.2.x component that prevented signal aggregation for signals stored in the cloud in Parquet format. For more information about the issue, see the Spark bug report *(https://issues.apache.org/jira/browse/SPARK-38318)*.

Fusion 5.7.0

Release date: December 13, 2022

Component versions:

Component	Version
Solr	fusion-solr 8.11.1 *(based on Solr 8.11.1)*
ZooKeeper	3.7.1
Spark	3.2.2
Kubernetes	GKE, AKS, EKS 1.24 Rancher (RKE) and OpenShift 4 compatible with Kubernetes 1.24 OpenStack and customized Kubernetes installs not supported. See *Kubernetes support* for end of support dates.
Ingress Controllers	Nginx, Ambassador (Envoy), GKE Ingress Controller Istio not supported.

More information about support dates can be found at *Lucidworks Fusion Product Lifecycle*.

	Looking to upgrade? Check out the *Fusion 5 Upgrades topic* for details.

Improvements

Fusion

- Fusion now supports Kubernetes 1.24. This applies to GKE, AKS, and EKS. It also applies to Rancher (RKE) and OpenShift 4 versions that are compatible with Kubernetes 1.24.
- Fusion now uses a custom Solr image called fusion-solr. This version of Solr is based on Solr 8.11.1 and adds functionality to support Fusion features.
- Upgraded Fusion to use Tika 2.4.1.
- Disabled browser autofill in Fusion data source configuration forms to prevent accidentally filling fields incorrectly.

Fusion Connectors

- The recrawl process is updated for the LDAP ACLs V2 connector. For more information, see *LDAP ACLs V2 connector incremental crawls*.
- As part of the transition from Pulsar to Kafka, the ability to *Remotely Launch a Plugin* is returned.
- A new connector, the *Web V2 Connector*, is added. This brings our Web connector from the classic framework to our V2 framework.
- The *Kaltura V2 Connector* is now available.

Predictive Merchandiser

- Improved UI elements in the Predictive Merchandiser.

Bug Fixes

Fusion

- Fixed an issue where pointing a Fusion collection at a Solr collection with a different name breaks queries to `solrServer` in Javascript stages.
- Fixed an issue when adding a new Graph security trimming stage; now you can enter a comma-separated list of datasources in **Exclude Data Source ID(s)** and **Inclusive Data Source ID(s)** fields with spaces. For example, `ds1, ds2, ds3`.
- Fixed an issue where documentation links pointed to Fusion 5.4 version, instead of the currently used version of Fusion.

Fusion Connectors

- The *LDAP ACLs V2 connector* is updated to reduce crawl bugs, prevent a false error from reporting when the crawl is stopped, and ensure permissions settings are kept in sync.
- Fixed an issue with network policies that prevented the Sharepoint Optimized V2 connector plugin from indexing some data.
- Fixed an issue with the ServiceNow connector where empty fields would cause an error and break the crawl.

Predictive Merchandiser

- Made the text box size larger to make it easier to enter multiple Boost Queries in the Rules Manager.

Known Issues

- A bug was discovered in the Spark 3.2.x component that prevents signal aggregation for signals stored in the cloud in Parquet format. For more information about the issue, see the Spark bug report *(https://issues.apache.org/jira/browse/SPARK-38318)*.
  This issue is fixed in Fusion 5.7.1.

Removals

- Removed deprecated NLP Annotator stage from Query Workbench and OpenNLP NER stage from Index Workbench.

Fusion 5.6.1

Release date: September 20, 2022

> Fusion 5.6.1 is available only to Managed Fusion customers.

Component versions:

Component	Version
Solr	8.11.1
ZooKeeper	3.7.1
Spark	3.2.1
Kubernetes	GKE, AKS, EKS 1.23 Rancher (RKE) and OpenShift 4 compatible with Kubernetes 1.23 OpenStack and customized Kubernetes installs not supported. See *Kubernetes support* for end of support dates.
Ingress Controllers	Nginx, Ambassador (Envoy), GKE Ingress Controller Istio not supported.

More information about support dates can be found at *Lucidworks Fusion Product Lifecycle*.

New Features

Fusion Connectors

- Fusion 5.6.1 adds a new connector, the *Web V2 Connector*. This brings our Web connector from the classic framework to our V2 framework.
  The Web V2 connector retrieves data from a Web site using HTTP and starting from a specified URL.
- As part of our transition from Pulsar to Kafka, 5.6.1 returns the ability to *Remotely Launch a Plugin*.

Bug Fixes

Fusion

- The *blocked rule form* now validates the "Parameter Name" when closing the modal. This fixes an error where blocked parameters in the Query Workbench could not be deleted via the UI.
- User with read-only access no longer see the *Quickstart* splash page, as they are unable to use it to create applications or other Fusion objects.

Fusion Connectors

- The *AWS S3 V2 Connector* connector is now able to crawl non-public buckets when the object key is not specified in the configuration.

Predictive Merchandiser

- Rules page now displays correctly for users with only the *developer role*.

Removals

Fusion SQL was removed as of Fusion 5.6.1. It is available for Fusion releases 5.5 and earlier.

Known issues

- A bug was discovered in the Spark 3.2.x component that prevents signal aggregation for signals stored in the cloud in Parquet format. For more information about the issue, see the Spark bug report *(https://issues.apache.org/jira/browse/SPARK-38318)*.

	This issue is fixed in Fusion 5.7.1.

- The link to the *Banana dashboards* in the Fusion UI does not work. You can access Banana dashboards using a direct link at `https://FUSION_HOST:FUSION_PORT/admin/banana/index.html#/dashboard/`.

	This issue is fixed in Fusion 5.7.0.

Fusion 5.6.0

Release date: August 10, 2022

	This release removes support for Pulsar. Any existing workflows using Pulsar will no longer function when upgrading to 5.6.

Component versions:

Component	Version
Solr	8.11.1
ZooKeeper	3.7.1
Spark	3.2.1
Kubernetes	GKE, AKS, EKS 1.22 Rancher (RKE) and OpenShift 4 compatible with Kubernetes 1.22 OpenStack and customized Kubernetes installs not supported. See *Kubernetes support* for end of support dates.
Ingress Controllers	Nginx, Ambassador (Envoy), GKE Ingress Controller Istio not supported.

More information about support dates can be found at *Lucidworks Fusion Product Lifecycle*.

	Looking to upgrade? Check out the *Fusion 5 Upgrades topic* for details.

New Features

Predictive Merchandiser

Predictive Merchandiser users can now fire templates and rules at every facet level.

Hierarchical template triggers allow you to fire a search configuration template or rules at a parent or child level in a hierarchy, and even at a faceted level.

Ways to use hierarchical templates:

- For a sale on "Winter Coats," you can fire a template at the "Outerwear" level or at any of its children.
- For an overstock on a particular clothing size, you could fire a template directly at the “Women’s Outerwear, Size Small” filter.
- When rolling out a new company policy, you could fire a zone on all pages from the homepage down so everyones sees the most important links and documents for the new policy.

See *Predictive Merchandiser Templates* for more information. For instructions on how to use Templates, see *Use Predictive Merchandiser Templates*.

Fusion

- Users can now manage API keys from their browser. See *API Keys* for more information.
- Added a new query pipeline stage, *Graph Security Trimming Stage*.
  The Graph Security Trimming stage restricts query results according to the user ID as an alternative to *Security Trimming Stage*. Whereas the Security Trimming stage has one Solr filter query per data source, Graph Security Trimming uses a single filter query for all data sources.

Fusion Connectors

- Added a new connector, the *SharePoint Optimized V2 Connector*. The SharePoint Optimized V2 connector retrieves content and metadata from an on-premises SharePoint repository. This connector replaces the SharePoint V2 Connector in Fusion 5.6.0.
- Added a new connector, the *LDAP-ACLs Connector*. This connector replaces the LDAP V2 Connector.

Bug Fixes

Fusion

- The Jobs panel no longer refreshes constantly while a job is running.
- The `fieldList` property in Index stages now displays the designated title and description.
- Fixed an issue that caused *Spark jobs* to fail when Fusion 5 was deployed using Gatekeeper service on Kubernetes.
- Configured *Realms* now maintain sign in information.
- In App Import, the **Import** button is no longer inactive if the wrong file type is selected.
- Fixed an issue that caused the *Job History API* to display inaccurate job status information.
- Fixed an issue that caused stage information to be deleted after moving them in the pipeline and saving.
- The `f5-monitoring-kube-state-metrics` service tooltip now displays.
- Fixed an issue that caused jobs to not run after adding a new schedule.
- Fixed an issue that caused an internal server error when creating a new app using numbers as an app name.
- Attempting to access the Rules page with a developer role no longer causes a forbidden error.
- Using the pipe symbol as a Field Delimiter in the Apply Rules query pipeline stage no longer causes a forbidden error.
- Fixed an issue that caused the timezone value to not save.

Predictive Merchandiser

- Fixed an issue that caused published templates to not display when clicking Start Task on the Merchandiser page.

Removals

- Removed **Selection of Solr Cluster** on the **Create New Collection** screen in the Admin UI.
- Removed support for Apache Pulsar and replaced it with Apache Kafka. As a result:
  - Logging via Pulsar has been removed and is replaced with a *preconfigured monitoring stack*.
  - The Log Viewer has been removed from the Admin UI.
  - *Remote connector support via Pulsar* is no longer possible. This functionality will be replaced in a future release.
  - The ability to access Pulsar from the Subscriptions API has been removed. You are no longer able to create a custom subscription topic.
  - The Send to Message Bus index stage has been removed.
- *LDAP-ACLs V2 Connector* replaces the *LDAP V2 Connector*.
- *SharePoint Optimized V2 Connector* replaces the *SharePoint V2 Connector*.

Known Issues

- *Helm chart deployments* will fail unless you pass the following values during the Helm install:

```
job-launcher: logstashEnabled: false job-rest-server: logstashEnabled: false
```

- A bug was discovered in the Spark 3.2.x component that prevents signal aggregation for signals stored in the cloud in Parquet format. For more information about the issue, see the Spark bug report *(https://issues.apache.org/jira/browse/SPARK-38318)*.
  This issue is fixed in Fusion 5.7.1.
- The link to the *Banana dashboards* in the Fusion UI does not work. You can access Banana dashboards using a direct link at `https://FUSION_HOST:FUSION_PORT/admin/banana/index.html#/dashboard/`.
  This issue is fixed in Fusion 5.7.0.

Fusion 5.5.2

Release date: May 31, 2022

Component versions:

Component	Version
Solr	8.11.1
ZooKeeper	3.6.2
Spark	3.2.1
Kubernetes	GKE, AKS, EKS 1.22 Rancher (RKE) and OpenShift 4 compatible with Kubernetes 1.22 OpenStack and customized Kubernetes installs not supported. See *Kubernetes support* for end of support dates.
Ingress Controllers	Nginx, Ambassador (Envoy), GKE Ingress Controller Istio not supported.

More information about support dates can be found at *Lucidworks Fusion Product Lifecycle*.

> Looking to upgrade?
> Check out the *Fusion 5 Upgrades topic* for details.

Improvements

- The Spark version has been updated to 3.2.1. See Spark Overview *(https://spark.apache.org/docs/latest/)* for more information.

Bug Fixes

- Fixed an issue that caused the ***click_signals_aggregation*** job to fail with a "Driver JVM" error.
- Fixed an issue that caused containers to crash when an Argo-based job failed to list the job status.

Known Issues

- A bug was discovered in the Spark 3.2.x component that prevents signal aggregation for signals stored in the cloud in Parquet format. For more information about the issue, see the Spark bug report *(https://issues.apache.org/jira/browse/SPARK-38318)*.

  > This issue is fixed in Fusion 5.7.1.

Removals

> The following features and functionalities are removed from default deployment in Fusion 5.5.2. See *Deprecations and Removals* for more information about suggested replacements:

- *Jupyter*
- *Superset*

Fusion 5.5.1–3

Release date: July 26, 2023

Component versions:

Component	Version
Solr	8.11.1
ZooKeeper	3.6.2
Spark	2.4.5
Kubernetes	GKE, AKS, EKS 1.25 and 1.26 Rancher (RKE) and OpenShift 4 compatible with Kubernetes 1.25 and 1.26 OpenStack and customized Kubernetes installs not supported. See *Kubernetes support* for end of support dates.
Ingress Controllers	Nginx, Ambassador (Envoy), GKE Ingress Controller Istio not supported.

More information about support dates can be found at *Lucidworks Fusion Product Lifecycle*.

	Looking to upgrade? Check out the *Fusion 5 Upgrades topic* for details.

Bug fixes

- Fixed an issue where Spark jobs failed when logging was used. You no longer need to use the workaround documented in the *5.5.1–1* and *5.5.1–2* release notes.
- Fixed an issue that prevented QnA Smart Answers jobs from finishing.
- Fixed an issue that prevented deploying Seldon models.
- Fixed an issue that prevented V2 connectors from running index pipelines configured with a Milvus stage.

Fusion 5.5.1–2

Release date: June 8, 2023

Component versions:

Component	Version
Solr	8.11.1
ZooKeeper	3.6.2
Spark	2.4.5
Kubernetes	GKE, AKS, EKS 1.25 and 1.26 Rancher (RKE) and OpenShift 4 compatible with Kubernetes 1.25 and 1.26 OpenStack and customized Kubernetes installs not supported. See *Kubernetes support* for end of support dates.
Ingress Controllers	Nginx, Ambassador (Envoy), GKE Ingress Controller Istio not supported.

More information about support dates can be found at *Lucidworks Fusion Product Lifecycle*.

A bug in the Netty Recycler component of fusion-spark can cause jobs to terminate early. To work around this issue, add the following parameters to the "Spark Settings" section:

```
spark.driver.extraJavaOptions=-
Dorg.apache.pulsar.shade.io.netty.recycler.maxCapacityPerThread=0
spark.executor.extraJavaOptions=-
Dorg.apache.pulsar.shade.io.netty.recycler.maxCapacityPerThread=0
```

For example:

Configuration to resolve Netty Recycler bug

This bug is fixed in *5.5.1–3*

Looking to upgrade?
Check out the *Fusion 5 Upgrades topic* for details.

New features

- Fusion 5.5.1–2 adds support for Kubernetes 1.25 and 1.26 to *Fusion 5.5.1*. Support for GKE is limited to Kubernetes 1.25, due to a GKE bug described in the *preceding note*.

Bug fixes

- Fixed a bug with the Tika parser that prevented Fusion from parsing Word documents added to the Blob Store.

Fusion 5.5.1-1

Release date: March 29, 2023

Component versions:

Component	Version
Solr	8.11.1
ZooKeeper	3.6.2
Spark	2.4.5
Kubernetes	GKE, AKS, EKS 1.24 Rancher (RKE) and OpenShift 4 compatible with Kubernetes 1.24 OpenStack and customized Kubernetes installs not supported. See *Kubernetes support* for end of support dates.
Ingress Controllers	Nginx, Ambassador (Envoy), GKE Ingress Controller Istio not supported.

More information about support dates can be found at *Lucidworks Fusion Product Lifecycle*.

A bug in the Netty Recycler component of fusion-spark can cause jobs to terminate early. To work around this issue, add the following parameters to the "Spark Settings" section:

```
spark.driver.extraJavaOptions=-
Dorg.apache.pulsar.shade.io.netty.recycler.maxCapacityPerThread=0
spark.executor.extraJavaOptions=-
Dorg.apache.pulsar.shade.io.netty.recycler.maxCapacityPerThread=0
```

For example:

Configuration to resolve Netty Recycler bug

This bug is fixed in *5.5.1-3*

Looking to upgrade?
Check out the *Fusion 5 Upgrades topic* for details.

Fusion 5.5.1-1 adds support for Kubernetes 1.24 to *Fusion 5.5.1*.

Fusion 5.5.1

Release date: April 13, 2022

Component versions:

Component	Version
Solr	8.11.1
ZooKeeper	3.6.2
Spark	2.4.5
Kubernetes	GKE, AKS, EKS 1.22 Rancher (RKE) and OpenShift 4 compatible with Kubernetes 1.22 OpenStack and customized Kubernetes installs not supported. See *Kubernetes support* for end of support dates.
Ingress Controllers	Nginx, Ambassador (Envoy), GKE Ingress Controller Istio not supported.

More information about support dates can be found at *Lucidworks Fusion Product Lifecycle*.

	Looking to upgrade? Check out the *Fusion 5 Upgrades topic* for details.

New features

- Merchandisers can now curate the metadata content of their pages with the **Page Metadata** feature in Predictive Merchandiser. See *Page Metadata* for more information.

Improvements

Fusion

- Added titles for Superset and Superset Proxy in the Fusion API **Microservices** dropdown menu.
- Improved logging from client-side retries, to make debugging `retry` jobs easier.
- Streamlined the **System** to remove redundant entries.

Predictive Merchandiser

- A script has been created to address the *`solrconfig.xml` workaround* listed in the Fusion 5.5.0 release notes, where users were directed to change their cache classes to `solr.search.CaffeineCache`. See the Fusion 5 Migration Guide *(https://github.com/lucidworks/fusion-cloud-native/tree/master/migrations#migration-from-fusion-5-4-x-to-5-5-x)* for more information.

Bug Fixes

Fusion

- Fixed an issue that caused the *Tika Parser* to generate "out of memory" errors.

Predictive Merchandiser

- Fixed an issue that caused images to not display in Predictive Merchandiser results and product details.
- Fixed an issue that caused the interface of the Create a facet console to display incorrectly after adding a new facet field.
- Fixed several UI issues that caused the Publish Rules dropdown to display incorrectly.

Fusion 5.4.5

Release date: December 17, 2021

Component versions:

Component	Version
Solr	8.8.2
ZooKeeper	3.6.2
Spark	2.4.5
Kubernetes	GKE, AKS, EKS 1.21 Rancher (RKE) and OpenShift 4 compatible with Kubernetes 1.21 OpenStack and customized Kubernetes installs not supported. See *Kubernetes support* for end of support dates.
Ingress Controllers	Nginx, Ambassador (Envoy), GKE Ingress Controller Istio not supported.

More information about support dates can be found at *Lucidworks Fusion Product Lifecycle*.

	Upgrade immediately! Check out the *Fusion 5 Upgrades topic* for details.

Fusion 5.4.5 addresses CVE-2021-44228 *(https://nvd.nist.gov/vuln/detail/CVE-2021-44228)*, a critical, zero-day exploit discovered in Apache Log4j, a logging tool used in many Java-based applications.

This widespread exploit affects all versions of Fusion and Solr. If Fusion and Solr are correctly secured behind a firewall and queries are sanitized by middle-ware, the risk of this vulnerability being exploited is low. However, as with all vulnerabilities, if an attacker gains access to the server, this vulnerability can be exploited.

The Fusion 5.4.5 release upgrades Log4j to 2.16.0 throughout Fusion. The only exception is for *Apache Pulsar*, a third-party multi-tenant messaging system, which is awaiting an official update. Although Pulsar in Fusion 5.4.5 uses Log4j 2.15.0, Pulsar is ***not susceptible*** to CVE-2021-44228 or the related CVE-2021-45046 *(https://nvd.nist.gov/vuln/detail/CVE-2021-45046)*, due to how the service is configured in Fusion.

	Security bulletin For detailed information on CVE-2021-44228 and its impact on Lucidworks products, see our security bulletin *(https://support.lucidworks.com/hc/en-us/articles/4415649244055-Log4J-zero-day-vulnerability-CVE-2021-44228)*.

Fusion 5.5.0

Release date: March 18, 2022

Component versions:

Component	Version
Solr	8.11.1
ZooKeeper	3.6.2
Spark	2.4.5
Kubernetes	GKE, AKS, EKS 1.22 Rancher (RKE) and OpenShift 4 compatible with Kubernetes 1.22 OpenStack and customized Kubernetes installs not supported. See *Kubernetes support* for end of support dates.
Ingress Controllers	Nginx, Ambassador (Envoy), GKE Ingress Controller Istio not supported.

More information about support dates can be found at *Lucidworks Fusion Product Lifecycle*.

	Looking to upgrade? Check out the *Fusion 5 Upgrades topic* for details.

Improvements

- Fusion 5.5 adds support for Rancher. See *Deploy Fusion 5 on Other Kubernetes Platforms* for more information.

Security

- Fusion services have been updated with the following versions:
  - Log4J service version 2.17.1
  - Tika 1.28
  - nginix 1.21.5-alpine

Bug Fixes

Fusion

- Fixed an issue that caused problems with access and configuration of kubelet options.
- Fixed an issue that potentially allowed Read roles to create/delete Solr operations.
- Fixed an issue that prevented the logout option from being displayed if the "logout" button was clicked while creating or modifying an OIDC/SAML *security realm*.
- Fixed an issue that caused the cursor to bypass the password field when using the tab button at login.
- Fixed an issue that prevented *App Insights* from creating custom reports.
- Fixed an issue that caused invalid language codes to be displayed in the ***Detect Language*** index pipeline stage.
- Fixed an issue that caused the *job list* to disappear after saving the job settings.

Predictive Merchandiser

- Fixed an issue that caused console and facet elements to stretch too far across the screen.
- Fixed an issue that made the **+ Add** button unavailable to associate products with *zero results*.
- Fixed an issue that caused results to disregard sorting attempts.
- Fixed an issue that caused ***facet*** values with child facets to not expand when clicked.
- Fixed an issue that prevented facet ordering from working as expected.
- Fixed an issue that caused analytics chart cards to remain blank.
- Fixed an issue that caused facets with no values to produce an error in query logs.

Connectors

- Various Connector improvements and bug fixes.

Removals

The following features and functionalities are removed in Fusion 5.5.0:

- The SharePoint V1 connector.
- Usage of the `with` keyword syntax in the *JavaScript Query Stage* has been removed, as usage can result in random errors. See MDN Web Docs *(https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/with)* for more information.
- Child facet values have been removed from *Hierarchical Facet* breadcrumbs in favor of displaying facet values, to avoid confusion.

Known Issues

Users may experience one of the following issues with Predictive Merchandiser within the Fusion 5.5 release:

- The ***Rules*** screen no longer loads in Predictive Merchandiser when importing an app after building from Fusion 5.5.
- After importing an app running an earlier version of Solr, newly-created facets do not appear in the *facet zone*.
  Workaround Steps:
  In the `solrconfig.xml`, all cache classes that use `solr.LRUCache` must be changed to `solr.search.CaffeineCache`. See Cache Implementations *(https://solr.apache.org/guide/8_7/query-settings-in-solrconfig.html#cache-implementations)* for more information.
- Changing the **Sort by** order of results is not reflected in the results list.

Fusion 5.4.4

Release date: December 2, 2021

Component versions:

Component	Version
Solr	8.8.2
ZooKeeper	3.6.2
Spark	2.4.5
Kubernetes	GKE, AKS, EKS 1.21 Rancher (RKE) and OpenShift 4 compatible with Kubernetes 1.21 OpenStack and customized Kubernetes installs not supported. See *Kubernetes support* for end of support dates.
Ingress Controllers	Nginx, Ambassador (Envoy), GKE Ingress Controller Istio not supported.

More information about support dates can be found at *Lucidworks Fusion Product Lifecycle*.

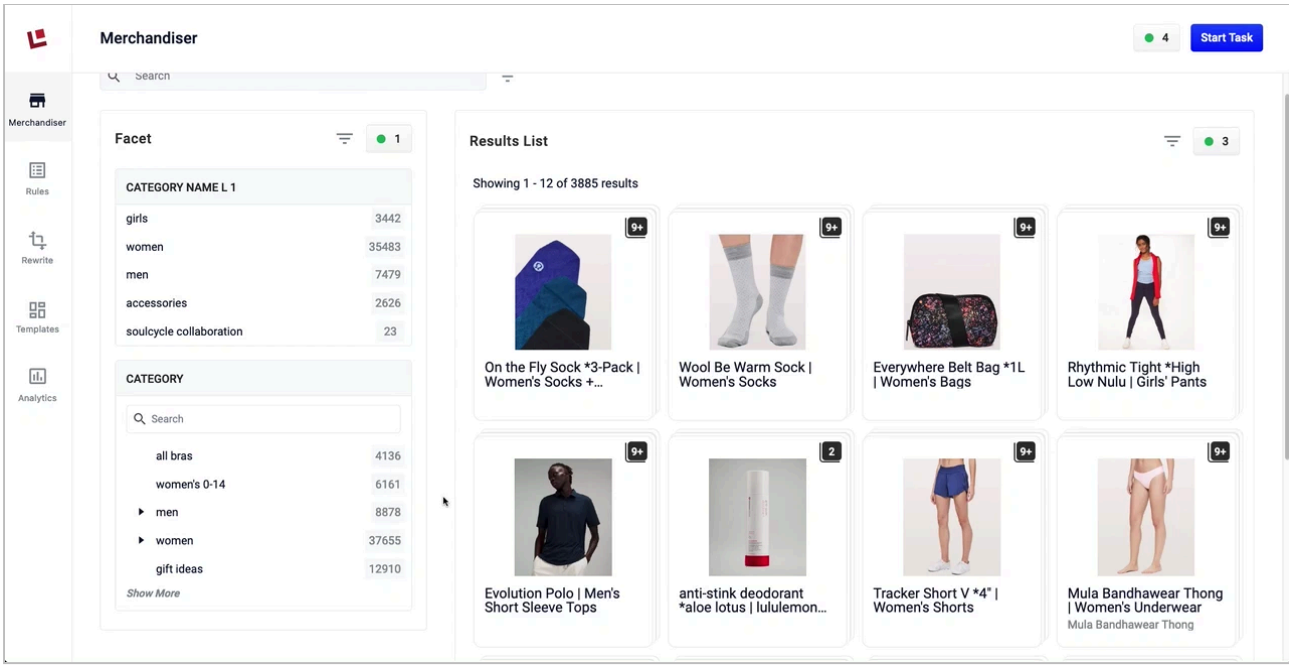
	Looking to upgrade? Check out the *Fusion 5 Upgrades topic* for details.

New features

Predictive Merchandiser

Hierarchical facets

Predictive Merchandiser facet zones can now use Fusion's *hierarchical facet* ability. Facet zones are organized with parent and child facets, allowing for faster and easier navigation as a user goes down and up the facet structure. Fields used for hierarchical facets are split according to the delimiter set in the *Apply Rules* query pipeline stage.



Improvements

Fusion

- An update to the LDAP realm expands group authorization capabilities by enabling subtree search. You can now use LDAP groups from more than one hierarchical parent.
- General UI improvements.

Bug fixes

Fusion

- In prior versions, certain jobs would fail if indexing was unresponsive with no clear failure reason. This release fails a job automatically after a certain period of unresponsiveness. The time to job failure is configurable. A failure reason also displays.
- Previously, Fusion indexing would register index pipeline failures as negative acknowledgements. This behavior caused unexpected retries. Now after a number of retries, indexing will display a success or an error only on the last attempt.
- A bug in the Fusion Index Pipeline UI covered the pipeline editing section when an index pipeline had many subscribers. This bug is fixed.
- Previously, delete queries were completed individually. Delete queries are now completed in batches to improve performance.
- A permissions bug in the Helm chart prevented certain roles from accessing `networking.k8s.io`. This release updates the Helm chart to fix this bug.
- In prior versions, stopping and starting an experiment could result in Spark parameters disappearing from the job configuration. This bug is fixed.

Fusion Connectors

- Due to a change in Solr, a bug sometimes caused an error to appear related to the *Salesforce connector* security trimming. This bug is fixed.
- A bug caused a NullPointerException error in the *Web V1 Connector* when form authentication was configured. This bug is fixed.
- A bug in the SharePoint V2 connector UI did not save the addition of SharePoint webs, lists, and libraries when including site collections. This bug is fixed.

- In prior versions, large crawls with the SharePoint V2 connector would time out and fail. This bug is fixed.
- In prior versions, deleting an item during a crawl before it is requested would return an empty list as the item response, potentially causing an IndexOutOfBoundsException error. This bug is fixed.
- Other miscellaneous Fusion Connectors bug fixes.

Fusion 5.4.3

Release date: September 29, 2021

Component versions:

Component	Version
Solr	8.8.2
ZooKeeper	3.6.2
Spark	2.4.5
Kubernetes	GKE, AKS, EKS 1.18 Rancher (RKE) and OpenShift 4 compatible with Kubernetes 1.18 OpenStack and customized Kubernetes installs not supported. See *Kubernetes support* for end of support dates.
Ingress Controllers	Nginx, Ambassador (Envoy), GKE Ingress Controller Istio not supported.

More information about support dates can be found at *Lucidworks Fusion Product Lifecycle*.

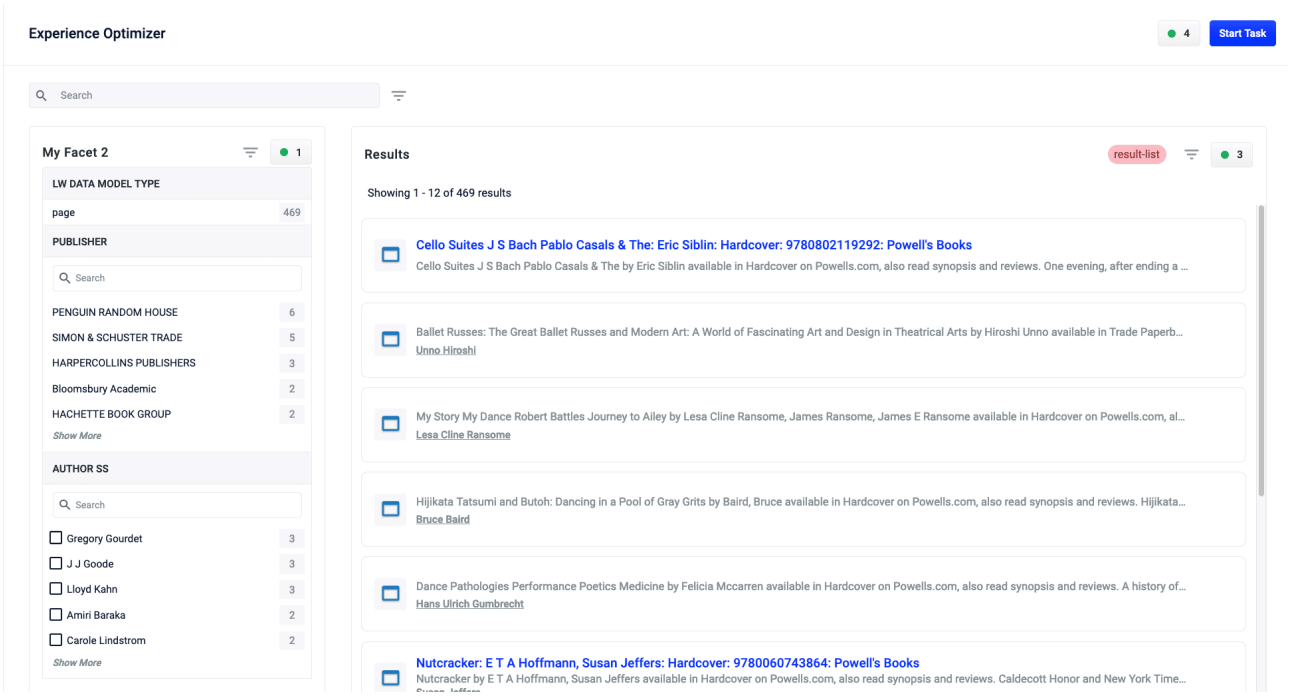
	Looking to upgrade? Check out the *Fusion 5 Upgrades topic* for details.

New features

Experience Optimizer

Fusion 5.4.3 introduces new solution for curating a powerful search experience: ***Experience Optimizer***. Similar to Predictive Merchandiser, Experience Optimizer is designed with knowledge management in mind. Knowledge managers can optimize search relevancy, create and manage rules, and analyze results without involving IT.

Knowledge and collaboration teams can now curate experiences and improve employee satisfaction, time-to-resolution, and R&D lead time. Users can detect synonyms or identify and fix low performing search terms. The visual analytics dashboard provides insight into opportunities for better conversion and allows knowledge managers to apply rules that pin, boost, bury, and block specific assets.



See *Licensing for Experience Optimizer* for licensing details.

Improvements

Fusion

- The LDAP security realm configuration includes a new option, **Use subtree scope?**. The subtree scope defines the search scope for all entries starting from the named object and all descendants below the named object.

Predictive Merchandiser

- The group items preview modal now shows more than 20 products at a time.
- When you update or delete a template or zone that has changed before you save your changes, Predictive Merchandiser returns a `409` conflict status.
- General UI improvements.

Bug fixes

- A bug is fixed that prevented rules from working as expected when the `id` field wasn't included in the **Return Fields** stage.
- A bug is fixed that sometimes resulted in a `400` error when viewing a details modal.

Fusion 5.4.2

Release date: June 17, 2021

Component versions:

Component	Version
Solr	8.8.2
ZooKeeper	3.6.2
Spark	2.4.5
Kubernetes	GKE, AKS, EKS 1.18 Rancher (RKE) and OpenShift 4 compatible with Kubernetes 1.18 OpenStack and customized Kubernetes installs not supported. See *Kubernetes support* for end of support dates.
Ingress Controllers	Nginx, Ambassador (Envoy), GKE Ingress Controller Istio not supported.

More information about support dates can be found at *Lucidworks Fusion Product Lifecycle*.

Docker image versions:

```
REGISTRY_HOST:PORT/lucidworks/admin-ui:5.4.2
REGISTRY_HOST:PORT/lucidworks/api-gateway:5.4.2
REGISTRY_HOST:PORT/lucidworks/auth-ui:5.4.2
REGISTRY_HOST:PORT/lucidworks/check-fusion-dependency:v1.2.0
REGISTRY_HOST:PORT/lucidworks/check-fusion-dependency:v1.3.0
REGISTRY_HOST:PORT/lucidworks/classic-rest-service:5.4.2
REGISTRY_HOST:PORT/lucidworks/connector-plugin:5.4.2
REGISTRY_HOST:PORT/lucidworks/connectors:5.4.2
REGISTRY_HOST:PORT/lucidworks/connectors-backend:5.4.2
REGISTRY_HOST:PORT/lucidworks/devops-ui:5.4.2
REGISTRY_HOST:PORT/lucidworks/fusion-api:5.4.2
REGISTRY_HOST:PORT/lucidworks/fusion-indexing:5.4.2
REGISTRY_HOST:PORT/lucidworks/fusion-log-forwarder:5.4.2
REGISTRY_HOST:PORT/lucidworks/fusion-spark:5.4.2
REGISTRY_HOST:PORT/lucidworks/insights:5.4.2
REGISTRY_HOST:PORT/lucidworks/job-launcher:5.4.2
REGISTRY_HOST:PORT/lucidworks/job-rest-server:5.4.2
REGISTRY_HOST:PORT/lucidworks/keytool-utils:v2.1.0
REGISTRY_HOST:PORT/lucidworks/ml-model-service:5.4.2
REGISTRY_HOST:PORT/lucidworks/query-pipeline:5.4.2
REGISTRY_HOST:PORT/lucidworks/rules-ui:5.4.2
REGISTRY_HOST:PORT/lucidworks/templating:5.4.2
REGISTRY_HOST:PORT/lucidworks/webapps:5.4.2
alpine:latest
apache/tika:1.25
apachepulsar/pulsar-all:2.6.3
argoproj/argocli:v2.7.6
argoproj/argoexec:v2.7.6
argoproj/workflow-controller:v2.7.6
bitnami/kubectl
busybox:1.31.1
datawire/ambassador:0.86.1
milvusdb/milvus:0.10.2-cpu-d081520-8a2393
mysql:5.7.30
seldonio/seldon-core-operator:1.1.0
solr:8.8.2
zookeeper:3.6.2
```

> Looking to upgrade?
> Check out the *Fusion 5 Upgrades topic* for details.

New Features

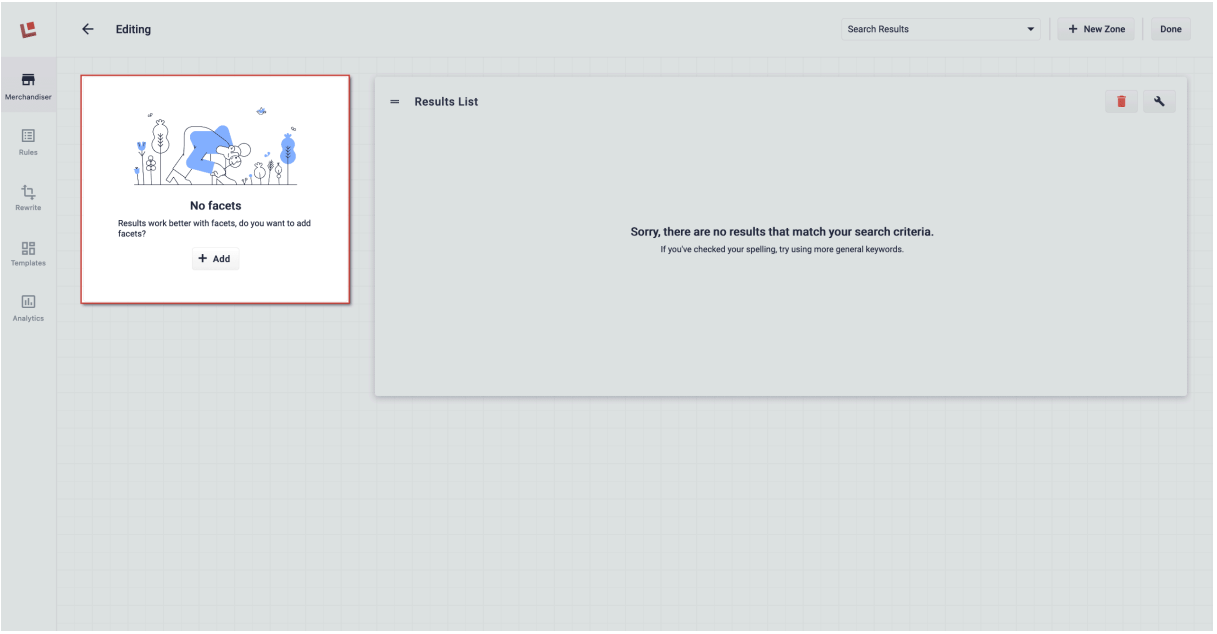
Custom Spark Jobs

Custom Spark Jobs are back. Removed in Fusion 4, this job is a viable alternative to *Custom Python Jobs* and Scala *Script Jobs*. Custom Spark Jobs allow you to simply upload a JAR file to the *blob manager* and specify a class name in the job configuration.

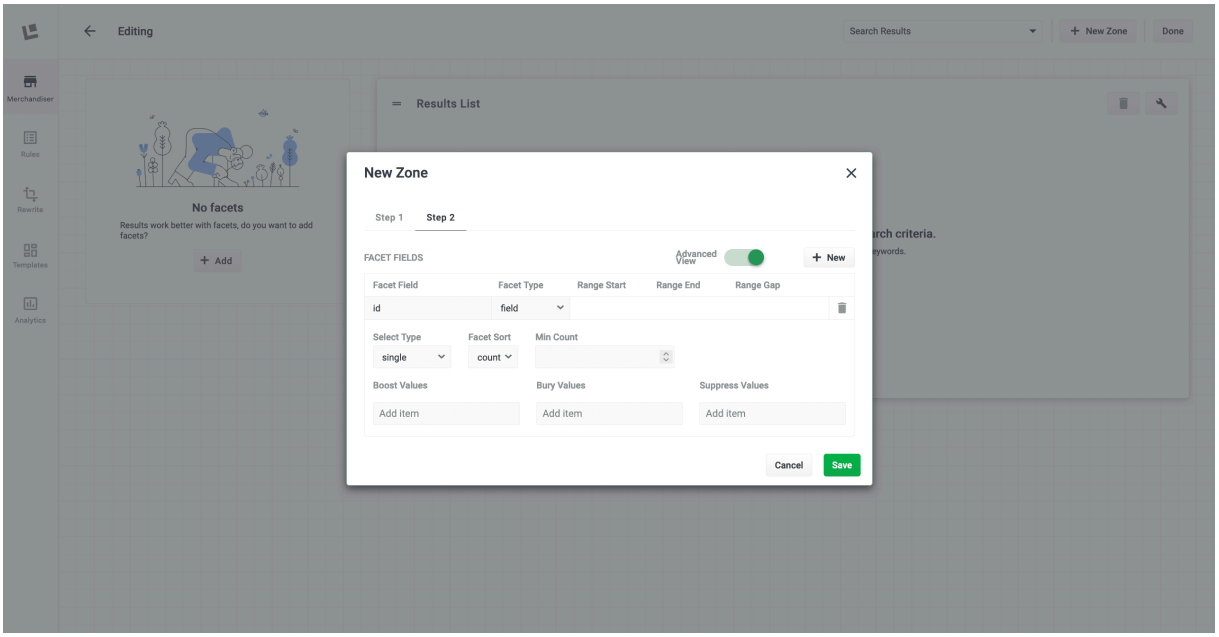
If you're upgrading from Fusion 4.x to 5.2.4+, and your search current application utilizes Custom Spark Jobs, this update may make your migration easier.

Improvements

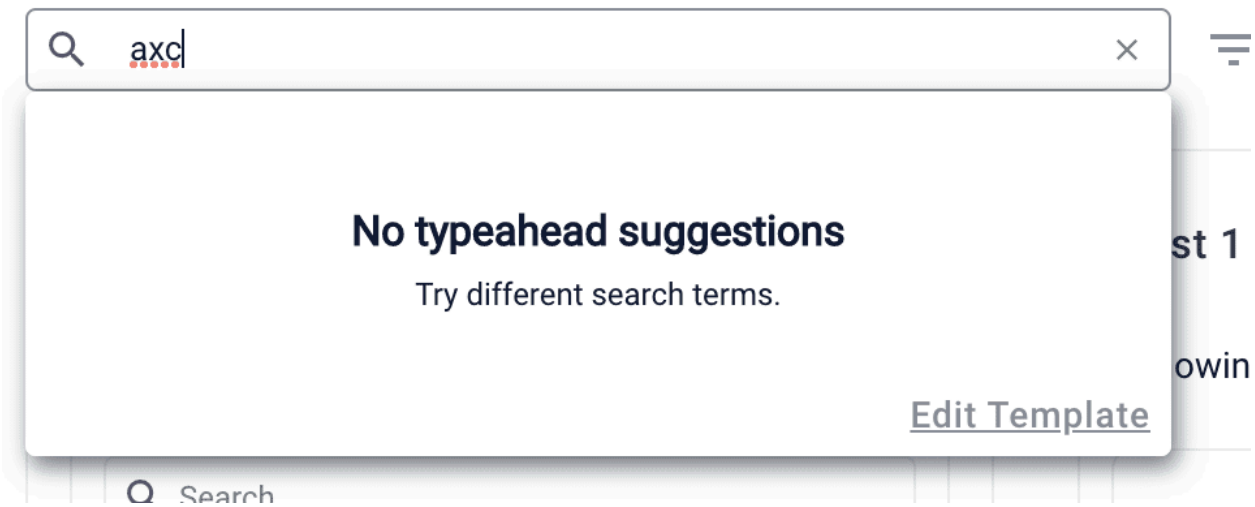
- The facet zone configuration form now allows you to create a facet zone with *Set Facets* rules:
  1. Create a facet zone in the **Templates** screen or directly in the **Predictive Merchandiser** screen.



2. Use the form to configure a custom facet zone with a **Set Facets** rule.



- When you enter a null results query in Predictive Merchandiser, the typeahead now gives a helpful notification instead of an empty results list.



Bug fixes

- The Confluence connector now redacts the password for the service account in the connector logs.
- A bug was fixed with the *Detect Language Index Stage* that could negatively impact downstream operations if the stage's RegEx operations became stuck. Now, the stage will time out if a field takes more than 10 seconds to resolve. The document is still indexed, but the language is not detected.
- A bug was fixed that required at least 3 fetch threads for the SharePoint Online connector to work. You can now set fewer than 3 fetch threads.
- A bug was fixed that prevented connector plugin service pods and backend service pods from starting if the Pulsar schema was unavailable.
- A bug was fixed that prevented authentication with an OIDC realm from working.
- A bug was fixed that prevented Fusion from importing a JSON object file if the JSON did not contain a `metadata` object on the same level as the `objects` object. For example:

  ```
  { "objects" : { "indexPipelines" : [ { "id": "abc-123", ... } ] }, "metadata": {} }
  ```

- Previously, jobs that timed out reported a "Success" status. Now, when a job times out, the error details message states:

  > Job terminated due to no plugin activity within *<number>* seconds.

- In Fusion 5.4.0, the JAX-RS libraries were upgraded, causing Appkit's webapps service to stop working as expected. The JAX-RS library upgrades are reverted to fix this issue.
- A bug was fixed in Predictive Merchandiser that removed the `)` character from the URL if the character appeared in a facet name, such as `(USA)`.
- A bug was fixed that sometimes prevented Predictive Merchandiser from marking zones as **Stale**.
- Miscellaneous UI bug fixes.

Fusion 5.4.1

Release date: May 6, 2021

Component versions:

Component	Version
Solr	8.8.2
ZooKeeper	3.6.2
Spark	2.4.5
Kubernetes	GKE, AKS, EKS 1.18 Rancher (RKE) and OpenShift 4 compatible with Kubernetes 1.18 OpenStack and customized Kubernetes installs not supported. See *Kubernetes support* for end of support dates.
Ingress Controllers	Nginx, Ambassador (Envoy), GKE Ingress Controller Istio not supported.

More information about support dates can be found at *Lucidworks Fusion Product Lifecycle*.

> Looking to upgrade?
> Check out the *Fusion 5 Upgrades topic* for details.

Bug fixes

- Added a missing dependency to the SQL microservice that is required for Pulsar.
- A bug was fixed in the connectors service that prevented indexing jobs from completing if an error occurred in the indexing streams. Now, errors are handled, and the job continues.

Known issues

- In Fusion 5.4.0, the JAX-RS libraries were upgraded, causing Appkit's webapps service to stop working as expected.

  > This issue is *fixed in Fusion 5.4.2*. Check out the *Fusion 5 Upgrades topic* for details on upgrading.

  Appkit apps deployed in Fusion 5.4.0 or 5.4.1 will not work, unless you downgrade the webapps microservice to 5.3.x. To do this:

  1. Open the `customize_fusion_values.yaml` file.
  2. Update the webapps image tag value to `5.3.5`, or any 5.3.x version number.

     ```
     webapps: image: tag: 5.3.5 livenessProbe: initialDelaySeconds: 60
     javaToolOptions: "-Xmx1g -Dspring.zipkin.enabled=false -
     Dspring.sleuth.enabled=false -XX:+ExitOnOutOfMemoryError" ...
     ```

  3. Save your changes.
  4. Run the upgrade script.

Fusion 5.4.0

Release date: April 23, 2021

Component versions:

Component	Version
Solr	8.8.2
ZooKeeper	3.6.2
Spark	2.4.5
Kubernetes	GKE, AKS, EKS 1.18 Rancher (RKE) and OpenShift 4 compatible with Kubernetes 1.18 OpenStack and customized Kubernetes installs not supported. See *Kubernetes support* for end of support dates.
Ingress Controllers	Nginx, Ambassador (Envoy), GKE Ingress Controller Istio not supported.

More information about support dates can be found at *Lucidworks Fusion Product Lifecycle*.

Solr is updated from version 8.6.3 to 8.8.2.

	Looking to upgrade? Check out the *Fusion 5 Upgrades topic* for details.

New Features

Fusion

Query Stage SDK
In addition to Fusion's *built-in query stages*, Lucidworks now provides a *Query Stage SDK* for developing your own custom query stages with Java. Resources are provided in a public GitHub repository *(https://github.com/lucidworks/query-stage-sdk)*.

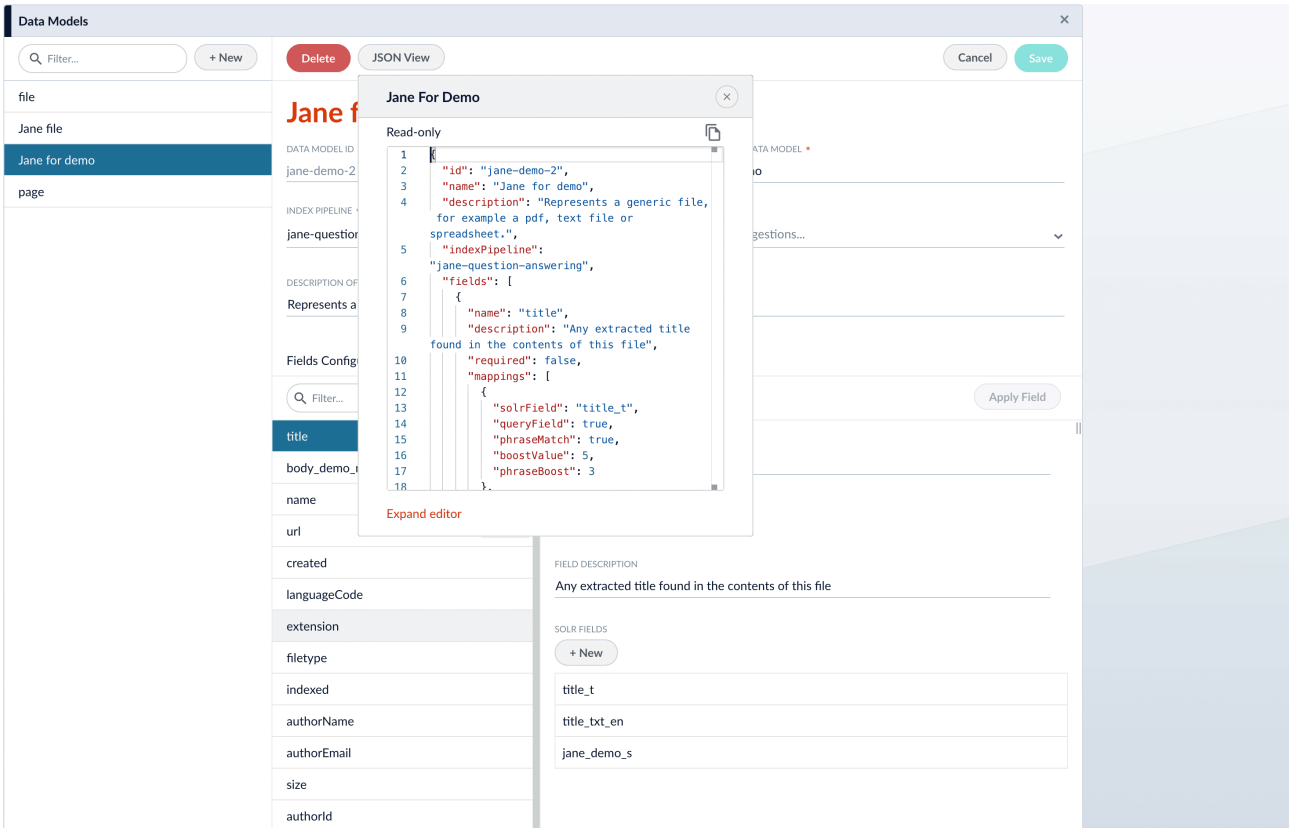
Solr request logs
A new Helm property, `solrRequestLogEnable`, enables the request log features of Solr when you set it to "true"; the default is "false".

Grouping Attributes
Grouping attributes can now be configured as part of the *Query Fields stage*. Previously, grouping attributes could only be configured in the *Additional Query Parameters stage*.

Editable JavaScript in the Fusion UI
JavaScript blobs can now be edited directly inside the Fusion UI using one of these interfaces:

- In the *blob manager* by clicking **Edit Blob**.
- In the *Managed Javascript index stage* or *Managed Javascript query stage*, by clicking **Edit** in the **Script Reference** field.

Pasteable data models in the Fusion UI
When *adding a new data model*, you can now paste it into the JSON view. Existing data models are read-only in the JSON view but their contents can be copied.



Superset
Apache Superset *(https://superset.apache.org/)* is now included as an optionally-deployable component of Fusion. See *Enable Superset (Fusion 5.4 and up)* for deployment instructions and *Use Superset with Fusion SQL* for a detailed tutorial.

TIKA Service (parser) Implementation
Apache Tika is now a *Fusion microservice* instead of an in-pod child process, for improved reliability and memory usage.

Text Processing Index stage
A new stage for the Index Pipeline, *Text Processing*, applies simple text transformations to text fields in a document.

Transfer Collection to Cloud job
The new *Transfer Collection to Cloud* job copies your Solr collections to cloud storage.

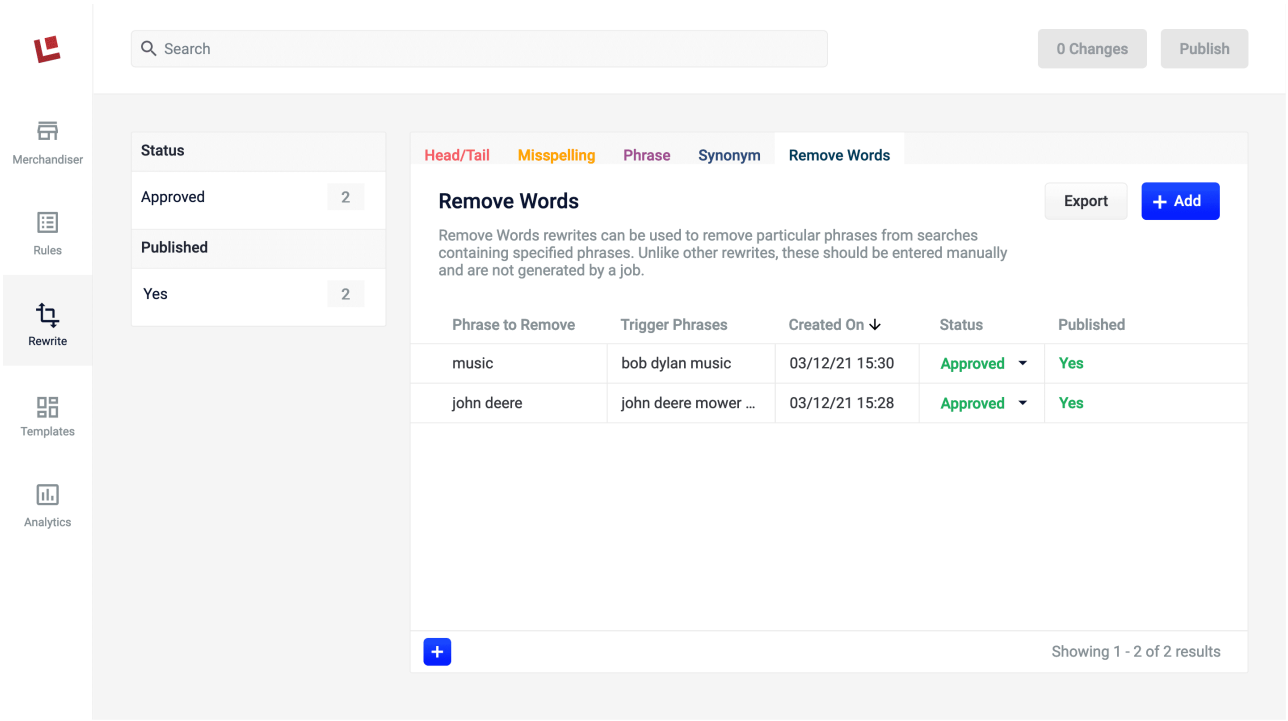
Improved language detection
The *Detect Language index stage* has new features enabled by these configuration keys:

- `languages` improves accuracy by specifying a limited set of candidate languages. Select the languages found in your content to prevent matching any other languages.
- `minimumConfidence` sets the minimum confidence score, to filter out low-confidence language matches.
- When `returnAllMatchedWithConfidenceScores` is enabled, the stage returns all languages whose confidence scores exceed the minimum, along with their corresponding confidence scores. Otherwise, only the language with the highest confidence score is returned.

Fusion AI

Remove Words rewrite type
A new rewrite type, *Remove Words*, is added to the Rules Editor's **Rewrite** screen.

Use a Remove Words query rewrite to remove particular phrases from queries. Unlike other rewrites, Remove Words rules are entered manually and aren't generated by a job.



To learn how to use Remove Words and other query rewrites in the *Rules Editor*, see *Use Query Rewrites in the Rules Editor*.

Data Augmentation job
The new *Data Augmentation* job provides various tasks designed to augment training and/or testing data, such as **Backtranslation**, **Split Word**, and **Synonym Substitution**.

Upload Model Parameters to Cloud job
Store the training blobs from your trained models in cloud storage with the new *Upload Model Parameters to Cloud* job.

Connectors

Configurable crawl fields for ServiceNow
The *ServiceNow connector* has a new `fields_to_crawl` property that defines the fields to be indexed per table; this also improves crawl performance.

Connectors SDK version 4.0.0

- A new caching interface is available in the *com.lucidworks.fusion.connector.plugin.api.cache* package.
- SDK-based connectors can now emit a date type using `LocalDateTime` in the *MapBuilder* class.
- *Skip* and *Error* items now accept metadata. Fields have been removed removed from `Skip` items; note that this will break existing connectors that use fields in `Skip` items.
- New packages: *com.lucidworks.fusion.connector.plugin.api.cache*, *com.lucidworks.fusion.connector.plugin.api.exceptions*, *com.lucidworks.fusion.connector.plugin.api.resource*.
- New classes: *BlobResource*, *BlobResourceClient*, *CacheManager*, *ConnectorPlugin.FetcherDef.Builder* *Content.ContentEmitter*, *ContentEmitException*, *FetchGroup*, *StaticSecuritySpec.UserPrincipalCase*.
- Some classes have been removed: `ContentFetcher.PostFetchContext`, `ContentFetcher.PreFetchContext`, `FetcherContext.AbstractPostFetchContext`, `FetcherContext.AbstractPreFetchContext`, `FetcherContext.DefaultPostFetchContext`, `FetcherContext.DefaultPreFetchContext`, `FetcherContext.PostFetchContext`, `FetcherContext.PreFetchContext`, `PostFetchResult`, `PreFetchResult`.

	Use `start()` and `stop()` in the Fetcher to replace the `preFetch()` and `postFetch()` methods that were previously deprecated and are now removed.

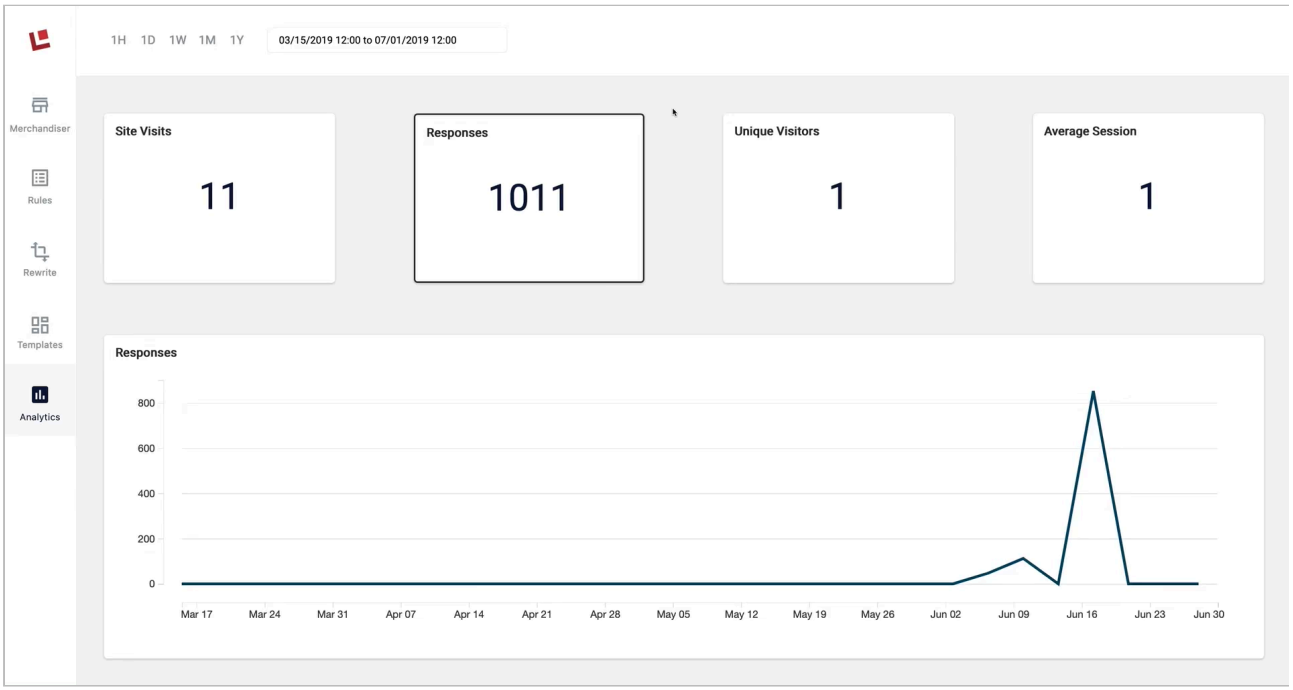
See the *Connectors SDK 4.0 Javadocs* for details.

Better labeling in the UI

In the Fusion UI, V1 and V2 connectors are now differentiated in the connectors list.

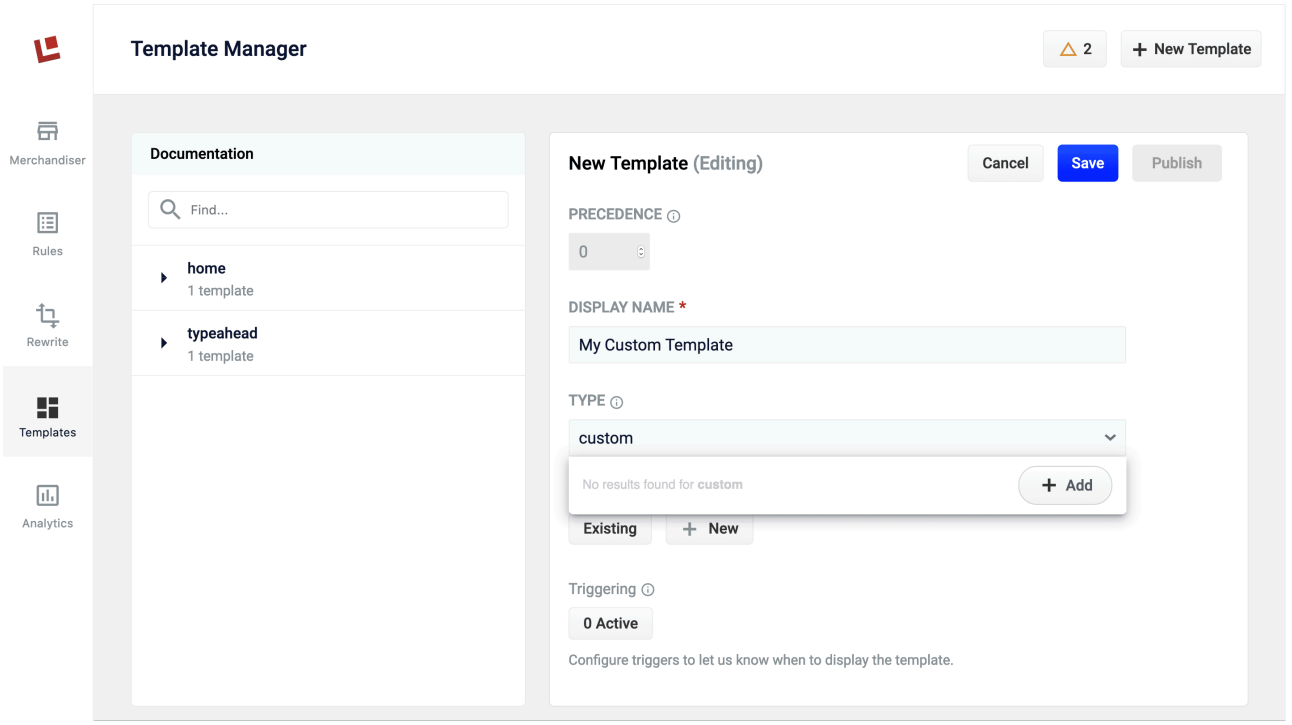
Predictive Merchandiser

Configurable Analytics
You can now configure the data displayed in the *Analytics screen* to highlight the signals most relevant to your needs.



To learn how, see *Use the Analytics screen in the Rules Editor*.

Custom Templates
The Rule Editor's *Templates* screen now allows you to *create custom template types*. This allows you to add more options for curating page types in your search application.



Improvements

Fusion

- The *Apache Pulsar subscription* type cannot be changed, but a subscription can be deleted and recreated with the desired type. This eliminates conflicts caused by a type change.
- Template ID and zone metadata were added to the *Templating API's* `render` endpoint. New `render` endpoints were also added to render templates by their specific template IDs.
- You can now configure the search mode (`all`, `legacy`, or `DSL`) for any *query profile* or *query pipeline*.
- *Search DSL* now supports grouping for facets.
- Indexing service logs now include `COLLECTION_NAME_lw_batch_id` to allow searching for errors associated with a specific connector job. Several additional stack tracing improvements were made.

Fusion AI

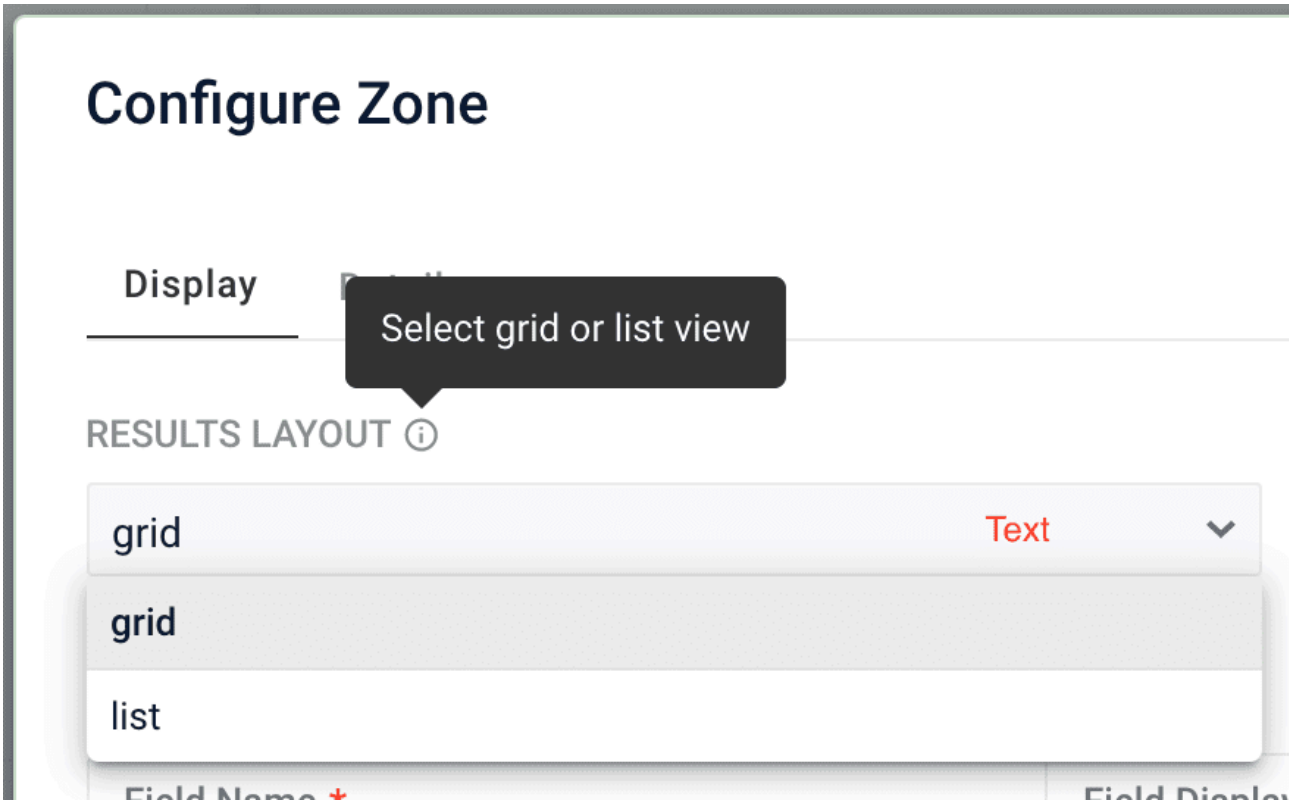
- The *BPR Recommender job* can now filter out low-interaction users with a new `minNumUserUniqueClicks` property.
- The default field names for *Trending Recommender jobs* now match the default names of the aggregated signals fields instead of raw signals fields.
- The default configuration values for the *Detect Language index stage* have been updated for better accuracy and fewer "unknown" language results.
- The Machine Learning Model Service now deletes models asynchronously for increased performance.
- The *Milvus Ensemble query stage* has a new `threshold` parameter that excludes items whose `ensemble_score` value is lower than the threshold. This is useful in your *Smart Answers pipeline configuration*.

Connectors

- The *SharePoint V2 connector* now indexes the metadata of files or attachments that do not meet the minimum or maximum size of file. Prior to the 5.4 release these files were ignored and not available for search.
- Pulsar messages now record the `statsResponse` from error messages.

Predictive Merchandiser

- The Request Time is now an independent field separated from the proxy logs message field. See the log viewer in *The DevOps Center* for Request Time and additional log information. You can now search Request Time independently for further monitoring, troubleshooting, and incident investigation.
- *Domain-Specific Language (DSL)* search is now compatible with Predictive Merchandiser.
- *Search rewrites* created by *Head/Tail Analysis* job are now editable, similar to synonyms.
- The limit on the number of boost/bury/suppress values in Facet Rules has been removed.
- Predictive Merchandiser templates are now organized in a functional tree. The functional tree allows templates to be easily found in the Template Manager.
- When *creating and editing zones*, you can now choose between grid and list layouts for results.



Bug fixes

- The JDBC V2 connector on longer ignores the Max Documents value.
- In the Resolve Multivalued Fields index stage UI, the `type` parameter was incorrectly labelled as "Field Name"; the label has been updated to "Type Name".
- In Predictive Merchandiser, pinned products now stay in position instead of moving to the top of the results.
- The JIRA Connector no longer throws an error when signatures are null.
- A bug was fixed that caused garbled Japanese characters when editing a blob in the Fusion UI.
- A bug was fixed that did not allow rules to trigger when filters include commas.
- A bug was fixed that caused additional fields in search results to disappear when many items are pinned in the Indexing Workbench.
- A bug that caused the V2 connector runtime to quickly run out of memory has been resolved.
- A bug was fixed that prevented highlighted results items using Domain Specific Language (DSL) syntax in the Query Workbench.
- A SharePoint connector security filtering bug, not allowing authorized files in a folder to be viewed, was resolved.
- Resolved known issues with SnakeYAML and Tomcat CVEs.
- A bug was fixed that prevented starting the gRPC service (`connectors-rpc`) when a Fusion deployment is behind a proxy.
- A bug was fixed that caused deleted values in the Partial Indexer stage to become empty strings.
- A bug was fixed that caused Active Directory (AD) V2 connector exceptions.
- A bug was fixed that caused the app import failure error "No such parsing configuration" with the SharePoint connector.
- Fixed a bug that generated a `PluginCall` error when the SharePoint V2 connector starts.
- A bug that could potentially delete documents when the document ID contains whitespace or special characters was resolved in the Access Control Collection (ACL) backend.

- A bug was fixed that caused issues in the XML Transformation index pipeline stage.
- The Indexing Service logs do not reveal database credentials.
- Some SharePoint crawl timeout issues were resolved.

Known issues

Fusion

- Validation errors occur when index pipelines are imported before data models. This issue is fixed in Fusion 5.4.2.
- OpenID Connect realms do not work using proxy settings. This issue is fixed in Fusion 5.4.2.
- After upgrading from 5.3.3 to 5.4.0, the **About Fusion** section displays the version incorrectly. This issue is fixed in Fusion 5.4.1.
- SP Plugin pods fail with an `OOMKilled` error.
- In Fusion 5.4.0, the JAX-RS libraries were upgraded, causing Appkit's webapps service to stop working as expected.

  > This issue is *fixed in Fusion 5.4.2*. Check out the *Fusion 5 Upgrades topic* for details on upgrading.

  Appkit apps deployed in Fusion 5.4.0 or 5.4.1 will not work, unless you downgrade the webapps microservice to 5.3.x. To do this:

  1. Open the `customize_fusion_values.yaml` file.
  2. Update the webapps image tag value to `5.3.5`, or any 5.3.x version number.

     ```
     webapps: image: tag: 5.3.5 livenessProbe: initialDelaySeconds: 60
     javaToolOptions: "-Xmx1g -Dspring.zipkin.enabled=false -
     Dspring.sleuth.enabled=false -XX:+ExitOnOutOfMemoryError" ...
     ```

  3. Save your changes.
  4. Run the upgrade script.

Removals

- The Slack connector has been removed. This is due to Slack removing a critical API endpoint used in the connector, `channels.*`. The *Slack V2 connector* is in development and will replace this connector.
- The option to view metrics not supported in 5.x is now hidden in the Dashboards (Banana).
- The `prefetch()` and `postfetch()` methods are removed from the Connectors SDK; now you use only `start()` and `stop()`.
- Index profiles no longer accept request parameters; these key/value pairs were never read or forwarded to the pipeline.
- The class passed into the last parameter of a JavaScript stage has changed from `SolrClientFactory` to `SolrClusterComponent`. This change only impacts users with advanced JavaScript stages.

Fusion 5.3.5

Release date: March 8, 2021

Component versions:

Component	Version
Solr	8.6.3
ZooKeeper	3.5.7
Spark	2.4.5
Kubernetes	GKE, AKS, EKS 1.19 Rancher (RKE) and OpenShift 4 compatible with Kubernetes 1.19 OpenStack and customized Kubernetes installs not supported. See *Kubernetes support* for end of support dates.
Ingress Controllers	Nginx, Ambassador (Envoy), GKE Ingress Controller Istio not supported.

More information about support dates can be found at *Lucidworks Fusion Product Lifecycle*.

	Looking to upgrade? Check out the *Fusion 5 Upgrades topic* for details.

Bug fixes

- Fixed a bug that sometimes prevented apps from importing into Fusion if an index pipeline referenced a parser used in the import file. Now, Fusion imports parsers before index pipelines.
- Fixed a bug with connectors that sometimes resulted in `deleteById` operations removing more documents than intended. Typically, this would occur if the `id` contained whitespace or special characters. This resulted from `deleteById` also executing `deleteByQuery` using SolrJ. Now, the operation uses `deleteById` only.

Known issues

Connectors

- When Form Authentication is configured, the Web Connector displays an error. This issue is fixed in Fusion 4.2.7 and 5.4.3.

Fusion 5.3.4

Release date: February 12, 2021

Component versions:

Component	Version
Solr	8.6.3
ZooKeeper	3.5.7
Spark	2.4.5
Kubernetes	GKE, AKS, EKS 1.19 Rancher (RKE) and OpenShift 4 compatible with Kubernetes 1.19 OpenStack and customized Kubernetes installs not supported. See *Kubernetes support* for end of support dates.
Ingress Controllers	Nginx, Ambassador (Envoy), GKE Ingress Controller Istio not supported.

More information about support dates can be found at *Lucidworks Fusion Product Lifecycle*.

Docker image versions:

```
lucidworks/admin-ui:5.3.4
lucidworks/api-gateway:5.3.4
lucidworks/auth-ui:5.3.4
lucidworks/check-fusion-dependency:v1.2.0
lucidworks/classic-rest-service:5.3.4
lucidworks/connector-plugin-service:5.3.4
lucidworks/devops-ui:5.3.4
lucidworks/fusion-api:5.3.4
lucidworks/fusion-indexing:5.3.4
lucidworks/fusion-log-forwarder:5.3.4
lucidworks/fusion-spark:5.3.4
lucidworks/insights:5.3.4
lucidworks/job-launcher:5.3.4
lucidworks/job-rest-server:5.3.4
lucidworks/keytool-utils:v2.1.0
lucidworks/ml-model-service:5.3.4
lucidworks/pm-ui:5.3.4
lucidworks/query-pipeline:5.3.4
lucidworks/rest-service:5.3.4
lucidworks/rpc-service:5.3.4
lucidworks/rules-ui:5.3.4
lucidworks/sql-service:5.3.4
lucidworks/templating:5.3.4
lucidworks/webapps:5.3.4
alpine:3.8
apachepulsar/pulsar-all:2.6.1
argoproj/argocli:v2.7.6
argoproj/workflow-controller:v2.7.6
bitnami/kubectl:1.15-debian-9
busybox:1.31.1
docker.io/seldonio/seldon-core-operator:1.1.0
milvusdb/milvus:0.10.2-cpu-d081520-8a2393
mysql:5.7.30
quay.io/datawire/ambassador:0.86.1
solr:8.6.3
zookeeper:3.5.7
```

> Looking to upgrade?
> Check out the *Fusion 5 Upgrades topic* for details.

New Features

Improvements

Connectors

- Updated *Salesforce* connector to accept `instanceUrl` as a configuration property, to better support OAuth flow.

Bug fixes

- Fixed a bug that caused the *Salesforce* connector to produce an error when retrieving refresh token.

Known issues

Fusion

- Importing an app fails citing parsing configuration, despite the parser configuration being in the `objects.json`. This issue is fixed in Fusion 5.3.5 and 5.4.

Connectors

- Connectors Security Trimming stage does not work in Solr nodes due to a change in Solr. This issue is fixed in Fusion 5.4.

Fusion 5.3.3

Release date: February 4, 2021

Component versions:

Component	Version
Solr	8.6.3
ZooKeeper	3.5.7
Spark	2.4.5
Kubernetes	GKE, AKS, EKS 1.19 Rancher (RKE) and OpenShift 4 compatible with Kubernetes 1.19 OpenStack and customized Kubernetes installs not supported. See *Kubernetes support* for end of support dates.
Ingress Controllers	Nginx, Ambassador (Envoy), GKE Ingress Controller Istio not supported.

More information about support dates can be found at *Lucidworks Fusion Product Lifecycle*.

> Looking to upgrade?
> Check out the *Fusion 5 Upgrades topic* for details.

The Fusion 5.3.3 release primarily addresses CVEs and known issues.

New Features

Connectors

- The *Confluence* connector now indexes the `createdDate` field in content from Confluence.

Improvements

Connectors

- Search results for `attachmentMaxSizeBytes` property now provide option to index metadata for large documents, or skip them. Definition of "large" size can also be specified.

Bug fixes

- Fixed a bug where Confluence indexed files with invalid extensions according to the Included and Excluded File Extensions lists.
- Removed stack traces from the Datasources API call responses.
- Fixed a bug where field mappings were not retained in web datasources.
- Fixed a bug that did not correctly convert the add, set, move, and copy operations in the field mapping stage.
- Fusion SQL code modified so assets backed by external Solr collections are not auto-deleted.

Known Issues

Fusion

- SP Plugin pods fail with an `OOMKilled` error.
- The XML Tranformation index stage outputs broken characters in individual document results. This issue is fixed in Fusion 5.4.

Fusion 5.3.2

Release date: December 14, 2020

Component versions:

Component	Version
Solr	8.6.3
ZooKeeper	3.5.7
Spark	2.4.5
Kubernetes	GKE, AKS, EKS 1.19 Rancher (RKE) and OpenShift 4 compatible with Kubernetes 1.19 OpenStack and customized Kubernetes installs not supported. See *Kubernetes support* for end of support dates.
Ingress Controllers	Nginx, Ambassador (Envoy), GKE Ingress Controller Istio not supported.

More information about support dates can be found at *Lucidworks Fusion Product Lifecycle*.

	Looking to upgrade? Check out the *Fusion 5 Upgrades topic* for details.

The Fusion 5.3.2 release primarily addresses CVEs and known issues.

Bug fixes

- Fixed a bug that prevented the creation of data models when the app name and app ID are different.
- Addressed reported CVEs in libraries used by various Fusion services.
- Fixed a bug that prevented the RPC service from working if the service was run from behind a proxy.
- Fixed a bug with the Machine Learning stage that resulted in authentication failing when using gRPC authentication headers. The ML library now passes JWT authentication when used as part of a Pulsar subscription as expected.
- Fixed a bug for the three parameters used in the *Smart Answers Supervised Training job*: `evalAnnIndex`, `useMixedPrecision`, and `useXbm`. This bug caused the `false` value to be ignored.

Fusion 5.3.1

Release date: December 11, 2020

Component versions:

Component	Version
Solr	8.6.3
ZooKeeper	3.5.7
Spark	2.4.5
Kubernetes	GKE, AKS, EKS 1.19 Rancher (RKE) and OpenShift 4 compatible with Kubernetes 1.19 OpenStack and customized Kubernetes installs not supported. See *Kubernetes support* for end of support dates.
Ingress Controllers	Nginx, Ambassador (Envoy), GKE Ingress Controller Istio not supported.

More information about support dates can be found at *Lucidworks Fusion Product Lifecycle*.

	Looking to upgrade? Check out the *Fusion 5 Upgrades topic* for details.

	Fusion 5.3.2 includes an important fix for a bug with data models. We recommend upgrading to Fusion 5.3.2.

Bug fixes

- Addressed reported CVEs in libraries used by various Fusion services.
- Fixed a bug that prevented the RPC service from working if the service was run from behind a proxy.
- Fixed a bug with the Machine Learning stage that resulted in authentication failing when using gRPC authentication headers. The ML library now passes JWT authentication when used as part of a Pulsar subscription as expected.
- Fixed a bug for the three parameters used in the *Smart Answers Supervised Training job*: `evalAnnIndex`, `useMixedPrecision`, and `useXbm`. This bug caused the `false` value to be ignored.

Known issues

Fusion

- The Partial Update Indexer stage sets multivalued fields to an empty string `""` instead of deleting them. *This issue* is fixed in Fusion 5.4.

Fusion 5.3.0

Release date: November 18, 2020

Component versions:

Component	Version
Solr	8.6.3
ZooKeeper	3.5.7
Spark	2.4.5
Kubernetes	GKE, AKS, EKS 1.19 Rancher (RKE) and OpenShift 4 compatible with Kubernetes 1.19 OpenStack and customized Kubernetes installs not supported. See *Kubernetes support* for end of support dates.
Ingress Controllers	Nginx, Ambassador (Envoy), GKE Ingress Controller Istio not supported.

More information about support dates can be found at *Lucidworks Fusion Product Lifecycle*.

Solr is updated from version 8.4.1 to 8.6.3.

	Looking to upgrade? Check out the *Fusion 5 Upgrades topic* for details.

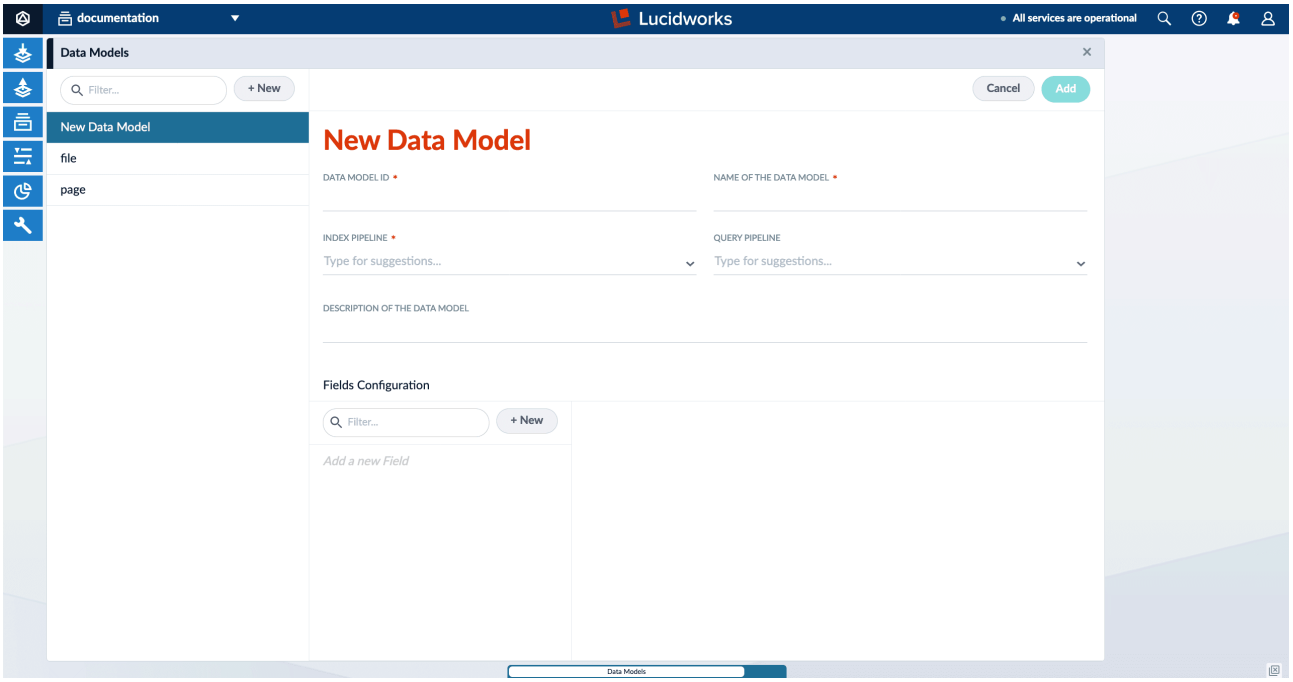
New Features

Fusion

Data Models

Data models simplify the process of getting started with Fusion by providing pre-configured objects to reduce the effort spent on basic starting tasks. This helps keep documents consistent between datasources and intuitive to the object's type.

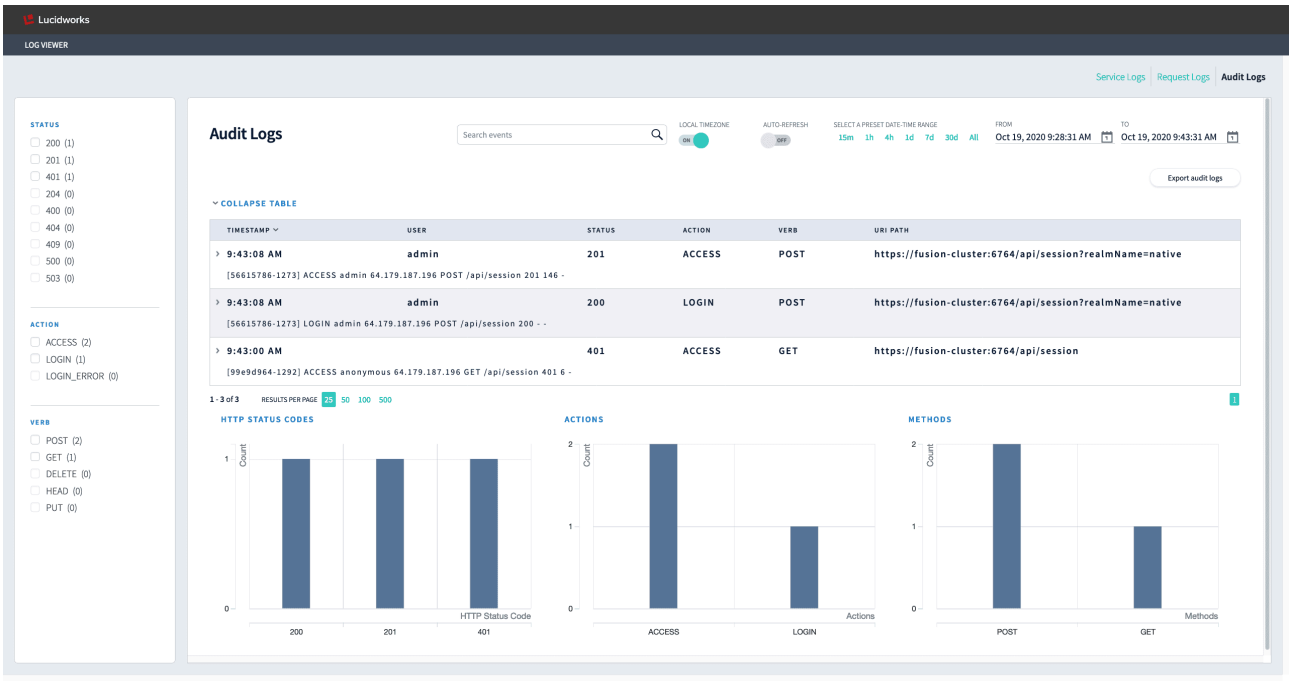
Access data models in the Fusion UI by navigating to **Indexing > Data Models**.



Some connectors include built-in data models as a standard component. Others require you to manually create data models.

Audit Logs

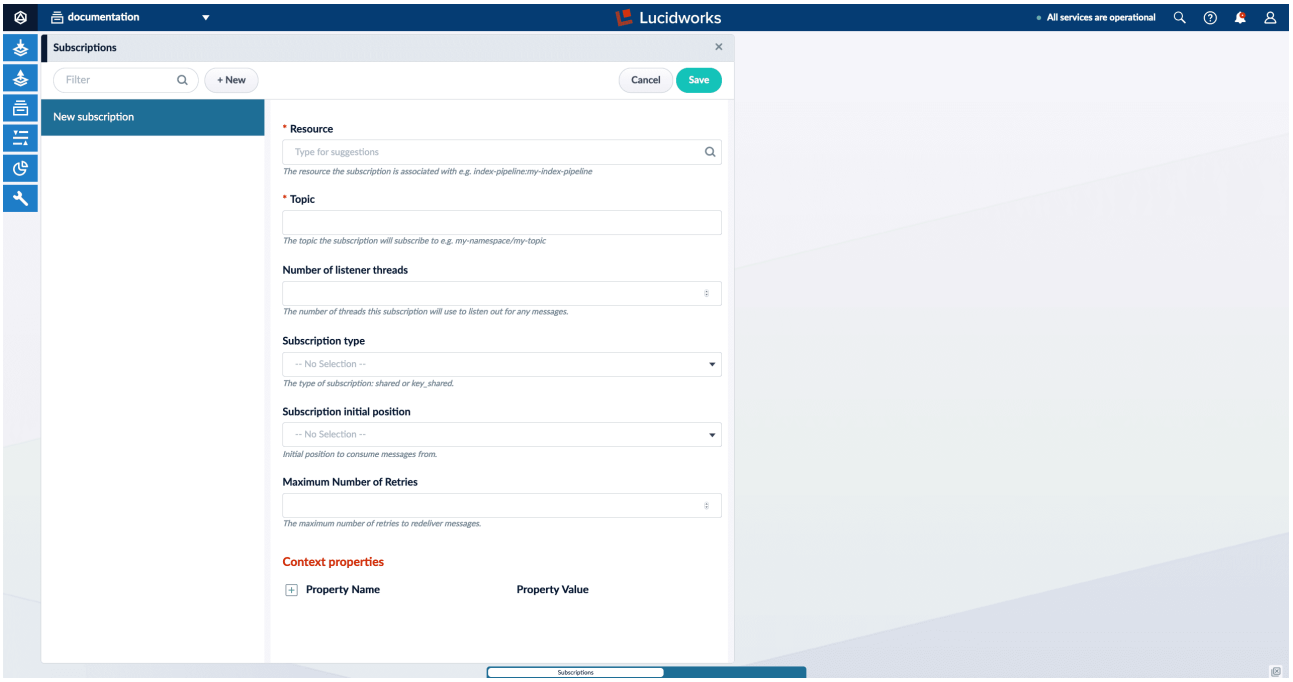
Audit Logs are added to the DevOps Center's Log Viewer. Audit logs provide you with a resource for tracking actions within Fusion, including the date, time, user responsible, and more.



Subscriptions

Subscriptions in Fusion allow you to create and configure subscriptions using *Apache Pulsar*.

In Fusion 5.3.0, Subscriptions are included in the Fusion UI under **Indexing > Subscriptions**.



See *Subscriptions UI* for more information.

Fusion

New features for Smart Answers

Milvus integration

Fusion 5.3 extends support for semantic search using vectors and embeddings by integrating with *Milvus*, a highly scalable embeddings engine that allows Fusion to streamline the methodologies that use deep learning for question/answer solutions like Smart Answers, recommendations based on similarity, and regular search. Milvus MySQL is included for metadata management.

A number of new components are introduced to manage and utilize Milvus:

New jobs
- *Create Indexes in Milvus*
- *Create Collections in Milvus*
- *Delete Indexes in Milvus*
- *Delete Collections in Milvus*

New pipelines
- *Smart Answers Index Pipeline*
- *Smart Answers Query Pipeline*
  For the latest configuration instructions, see *Configure The Smart Answers Pipelines*.

New index pipeline stage
- *Encode into Milvus*

New query pipeline stages
- *Milvus Ensemble Query*
- *Milvus Query*
- *Milvus Response Update*

New deep learning models

In Fusion 5.3, we are refreshing our deep learning models methodologies to be used in training and inference for semantic search-based Smart Answers. The following models are new in this release:

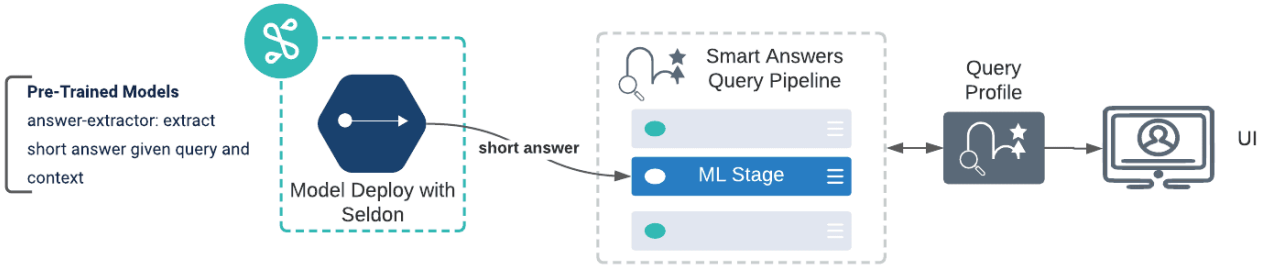
• bpe_en_300d_10K • bpe_en_300d_200K • bpe_ja_300d_100K • bpe_ko_300d_100K • bpe_zh_300d_50K • pe_multi_300d_320K	The bpe_{language}_{dim_size}_{vocab_size} models are general pre-trained BPEmb *(https://nlp.h-its.org/bpemb/)* embeddings that are available for different languages, including Chinese/Japanese/Korean (CJK) languages and multilingual. These are also useful in scenarios when vocabulary is very big or when the data might contain a lot of misspellings.
• distilbert_en • distilbert_multi	These are distilled, performance-optimized versions of BERT models *(https://arxiv.org/abs/1910.01108)* designed to be used on scale. Available for English language and multilingual applications.
• biobert_v1.1	This is a BERT model *(https://arxiv.org/abs/1901.08746)* that was pre-trained on large-scale biomedical corpora which makes it more suitable for biomedical domain applications.

For configuration details, see *Train A Smart Answers Supervised Model*, *Train A Smart Answers Cold Start Model*, and *Advanced Model Training Configuration for Smart Answers*.

Answer Extraction

To enhance how our Smart Answers customers interact with results sets that are composed of large documents, Fusion 5.3 adds Answer Extraction, allowing you to extract a paragraph, sentence, or phrase to answer questions.

When a large document is presented as a result to a query, Answer Extraction extracts the sentences out of the document that are most similar to the query content. To configure this feature, you train a model that gets deployed at the end of the Smart Answers query pipeline stage, after the resulting set of large documents is returned from Solr for final ranking. The model outputs the sentences from each document that are the most similar to the query.



The Answer Extraction model is now available from the Lucidworks official Docker *(https://hub.docker.com/r/lucidworks/answer-extractor)* to be deployed as a Seldon model. See *Extract Short Answers from Longer Documents* for detailed configuration steps.

New Seldon model: spaCy

The spaCy NER and POS model that formerly shipped with Fusion is now available only from the Lucidworks official Docker *(https://hub.docker.com/r/lucidworks/spacy-seldon)* to be deployed as a Seldon model.

The new Seldon model is compatible with Fusion 5.1+ and existing NLP Annotator stages.

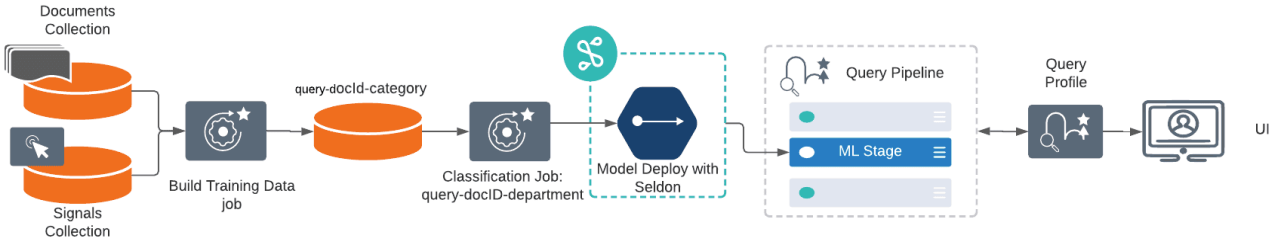
Trending Recommender job

The new *Trending Recommender job* analyzes signals to measure customer engagement over time. Use this job to identify spikes in popularity for specific documents or products, then display those items to your users or analyze the trends for business purposes. You can configure any time window, such as daily, weekly, or monthly.

For detailed steps to configure this job, see *Identify Trending Documents or Products*.

Build Training Data job

The new *Build Training Data job* constructs the training data required for query-time classification, that is, predicting the categories most likely to satisfy a query.



For detailed configuration steps, see *Classify New Queries*.

Connectors

Connectors SDK 3.0

Version 3.0 of the *Java Connector Development* is now available. See the *Javadocs* for complete details.

	Custom connectors created with previous versions of the SDK must be recompiled with version 3.0 for compatibility with Fusion 5.3.

Remote connector support

Remote connector support returns in Fusion 5.3.0. See *Use a Remote Connector with Pulsar Proxy* for more information.

Windows Share SMB 2/3 (V2)

The Windows Share connector can access content in a Windows Share or Server Message Block (SMB 2 and 3 protocols)/Common Internet File System (CIFS) filesystem.

For more information, see the *Windows Share SMB 2/3 V2 connector* reference documentation.

Google Drive (V2)

Unresolved directive in <stdin> - include::/fusion-connectors/reference/googledrivev2.asciidoc[tag=intro]

For more information, see the *Google Drive* reference documentation.

JDBC (V2)

Connect to any JDBC database.

For more information, see the *JDBC V2 connector* reference documentation.

AWS S3 (V2)

The AWS S3 V2 connector crawls items in a single bucket. You must specify the **bucket name** and **AWS region** in which that bucket is located.

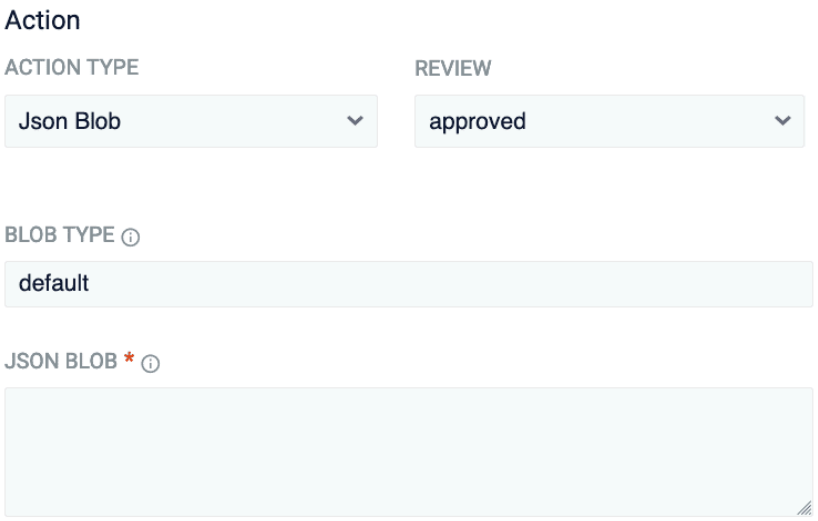
You may crawl specific items in a bucket. If no items are specified, the entire bucket will be crawled.

For more information, see the *AWS S3 V2 connector* reference documentation.

Predictive Merchandiser

JSON Blob rule type

A new rule type, *JSON Blob*, is added. This rule type allows you to pass arbitrary JSON blobs to your frontend when a rule fires:



Detail Page template

A new template, Detail Page, is added. This template allows you to configure what details and zones are displayed when a user views a product's details. To configure this template, navigate to **Templates** and edit **Detail Page**. You can also configure this template visually in the **Merchandiser** screen by hovering over a product, clicking the **Detail Page** button, clicking the **Start Task** button, and clicking the **Edit Template** button.

Improvements

Fusion

- Argo-based jobs can now be configured to access cloud storage. See *Configure An Argo-Based Job to Access GCS* and *Configure An Argo-Based Job to Access S3*.

- A new option is added to enable ephemeral users on the following *realms*: OpenID Connect, SSO Trusted HTTP, JWT, LDAP, and SAML. This improves performance for Fusion deployments with a large number of user accounts. To enable, set `ephemeralUsers` to `true`.

  > Ephemeral users are not created in Zookeeper, so the `autoCreateUsers` setting has no meaning if `ephemeralUsers` are enabled. The information needed to create a user must be taken from realm configuration and from IdP.

- V2 connector configurations are now persisted with default values.
- Improved error messages for Fusion SQL.
- Index and query stage schema is now returned alphabetically from `/api/index-stages/schema`.
- String templates are now supported in REST stages and collection parameters for the rules and text tagger stages.

Fusion UI

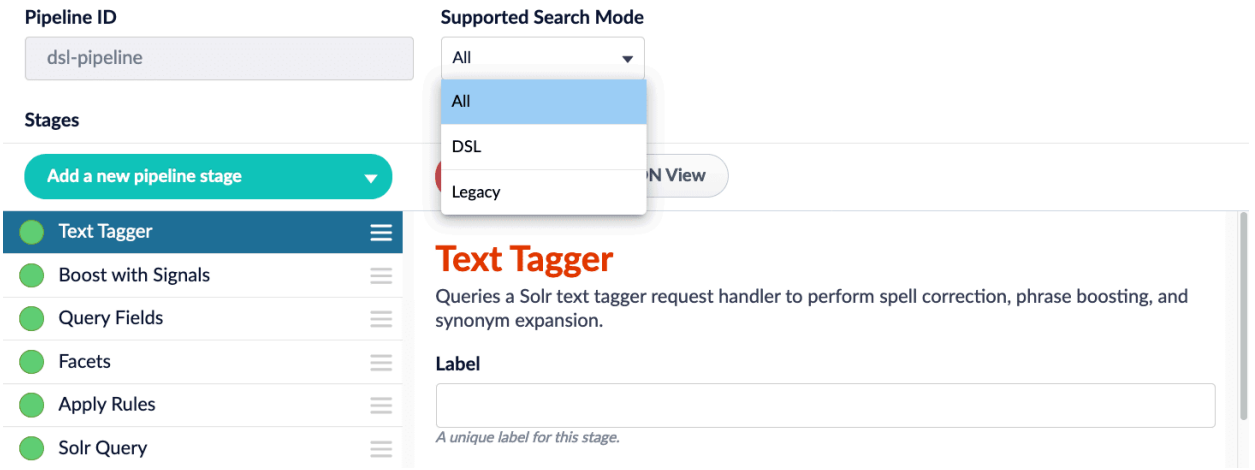
In addition to minor improvements, several notable improvements are made to the Fusion UI.

Fusion DSL integration

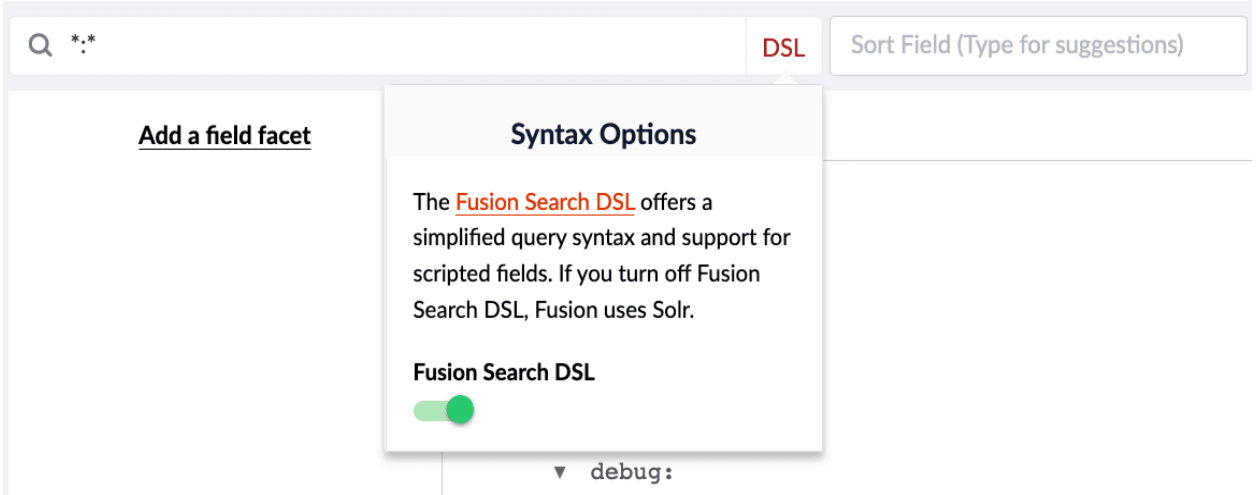
Fusion Domain Specific Language (DSL) provides expected search results as a JSON response in a way that reduces search query complexity for the user.

In Fusion 5.3.0, DSL is integrated with the UI:

- **Query Pipelines**. Select which search modes to use with your query pipeline: **DSL**, **Legacy** (Solr), or **All** (both). Your selection is reflected in the Query Workbench.

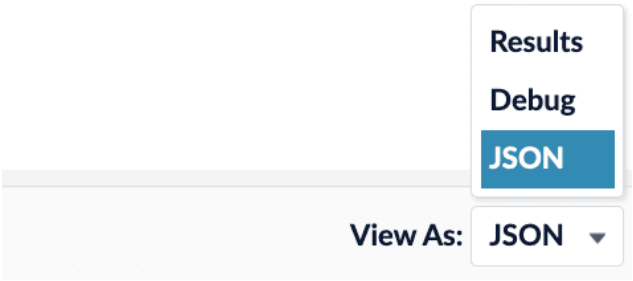


- **Query Workbench**. While entering queries in the Query Workbench, click the active search mode to toggle between DSL and Solr.

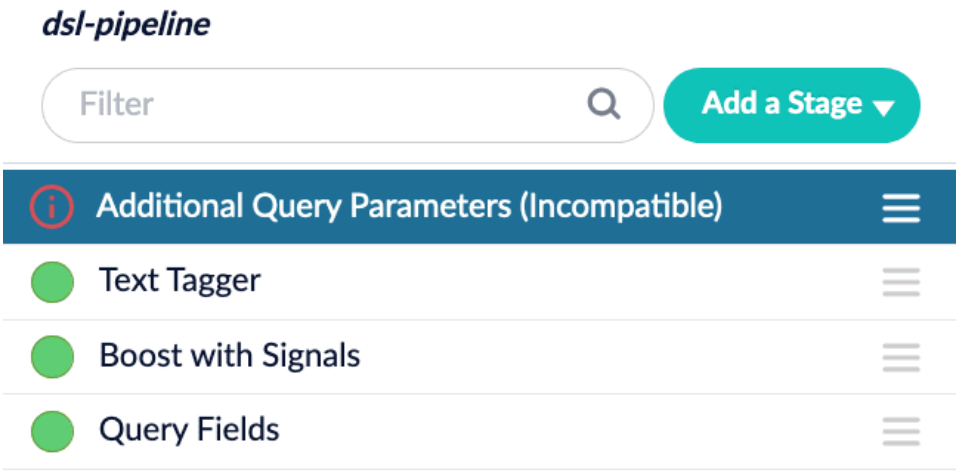


> This option is only available if your pipeline supports **All** search modes.

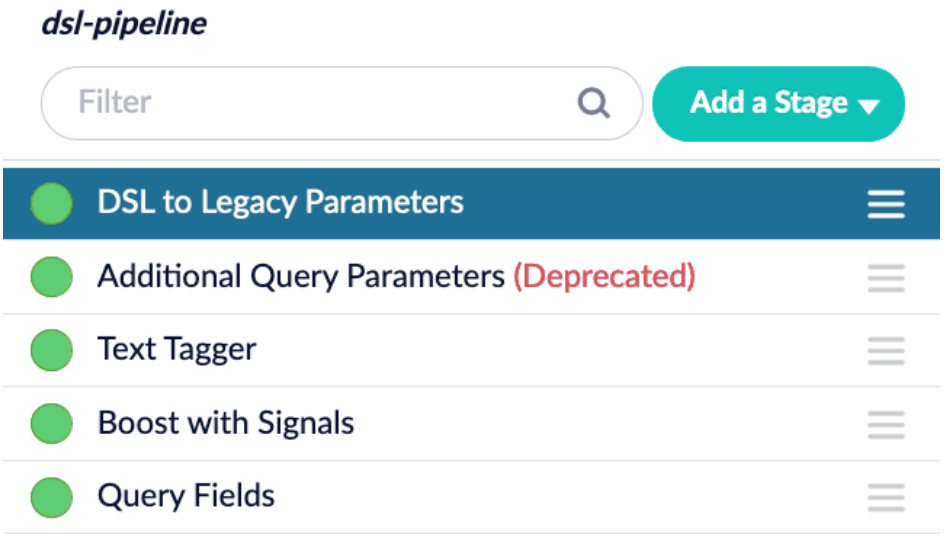
Change your **View As** setting to **JSON** to how the response results change.



- **Query Pipeline Stages**. Not all query pipeline stages support DSL. If you add an incompatible stage while in DSL mode, the stage will be flagged accordingly:



However, a new stage is added to extend DSL support to incompatible stages. Click **Add a Stage** and select the **DSL to Legacy Parameters** stage. When loaded before the incompatible stage(s), this stage converts the DSL request to legacy parameters so it can still be processed by the incompatible stage.



See the *DSL documentation* for more details on what you can do.

Minimap

A minimap is added to the bottom of the Fusion UI, allowing you to quickly move between open panels:

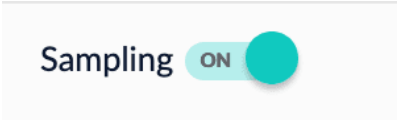
- Click one of the items to move your focus to the corresponding panel
- Close panels by hovering over an item and clicking the **X** that appears
- Close all panels by clicking the **Close all** button:

Panels with unsaved changes will not be closed.

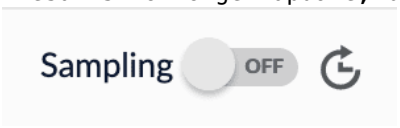
Toggle Sampling in the Index Workbench

The Index Workbench now features a **Sampling** toggle at the bottom of the results.

- When set to **On**, the results update with every change you make:

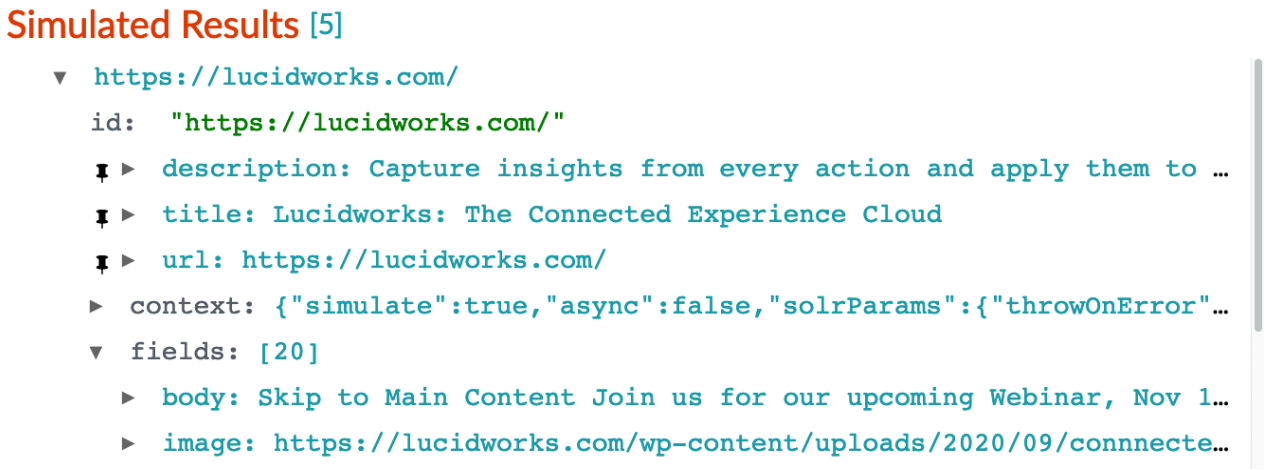


- When set to **Off**, the results no longer update, but a **Refresh** button appears to manually update the results:



Pin result fields

You can now pin result fields in the Index Workbench by hovering over a field and clicking the **Pin** icon that appears.



Pinned results are moved to the top of the document fields list, allowing you to quickly find the fields most relevant to the tweaks you are working on.

Fusion

- You can now view, edit, publish, and delete unpublished rules created by other users in the Rules Editor.
- AI jobs can now read and write from GCS directly. Meaning, the customer engagement data aka signals do not necessarily have to be stored in Solr for the jobs to work. This development is primarily to avoid situations where Solr is unable to keep up with the write requests from the jobs. GCS handles high speed, high scale writes efficiently.

Fusion SQL

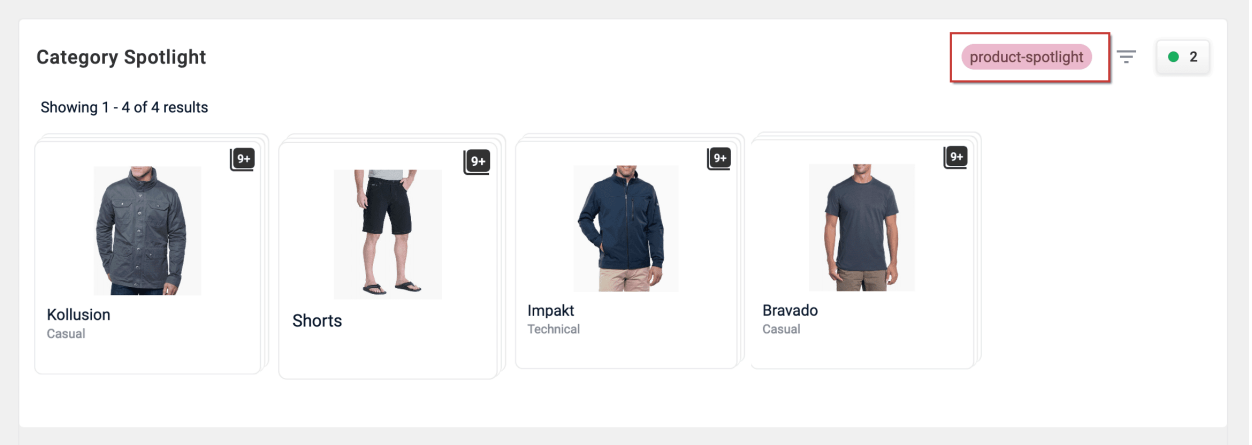
Enhancements in the Fusion SQL integration for notebooks and visualization platforms:

- **Jupyter**. Learn about connectivity set-up and usage examples of Fusion SQL with Jupyter notebooks in *Use Jupyter with Fusion SQL*.
- **Apache Superset**. Available documentation for connectivity set-up and usage of Fusion SQL with SuperSet BI platform in *Use Superset with Fusion SQL*

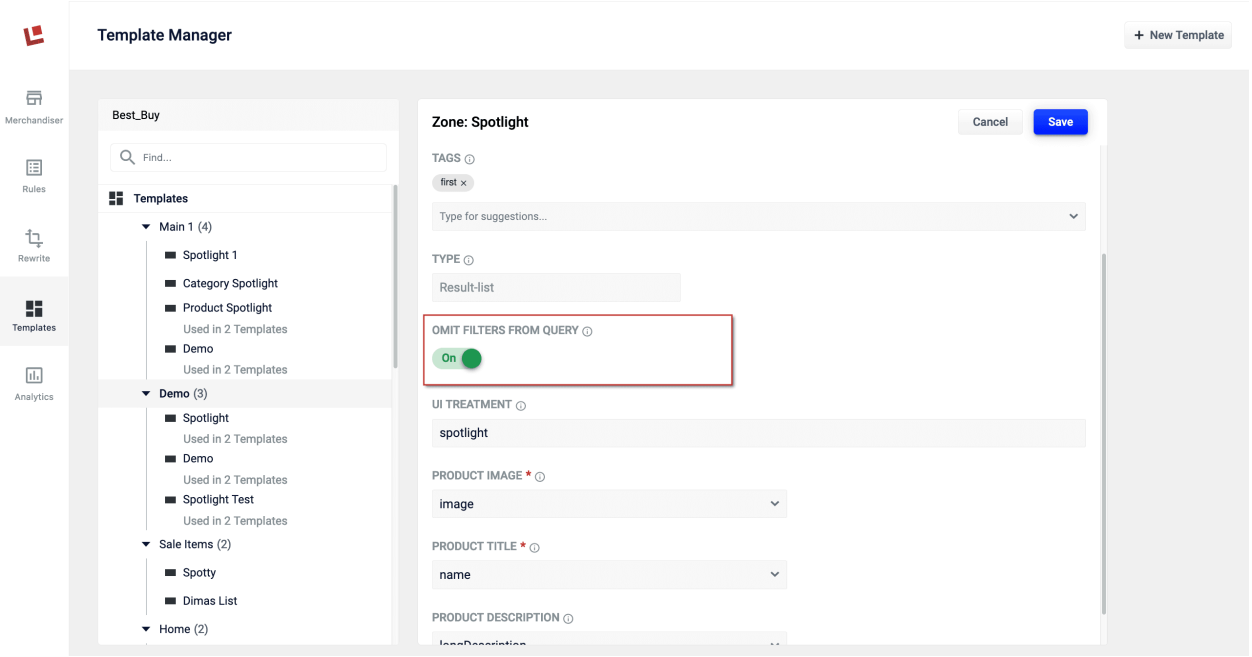
Predictive Merchandiser

- Predictive Merchandiser templates now support staging and publishing actions similar to rules. This allows you to draft templates and zones without affecting the production experience.

- Predictive Merchandiser zones adds a new configuration option, **UI Treatment.** This field allows for arbitrary text in the response when rendering this zone. This text is used by your frontend, when configured to do so, to determine how the zone is displayed. By utitilizing this field, you can use the same zone more than once on the same page but display the results differently.



- Predictive Merchandiser zones adds a new configuration option, **Omit Filters From Query.** When set to `true`, this zone ignores all `fq` filter parameters. For example, if you have a category landing page that is filtering results by category, this configuration option allows you to display items outside of that category.



- When attempting to navigate away from an unsaved template, a confirmation modal is now displayed to help prevent the loss of progress.
- Miscellaneous UI improvements.

Bug fixes

Fusion

- You can now easily select text in the log view without inadvertently collapsing the row.
- You can no longer save invalid field mapping configurations. Errors are now reported for invalid configurations.
- Fixed a bug that prevented jobs from saving multiple times.
- Fixed incorrect error messages in the Blob Store.

- Fixed a bug that sometimes prevented *Query-to-Query Session-based Similarity* jobs from saving.
- Reduced the number of requests sent to the Kubernetes API service to reduce the chance of exceeding the Kubernetes call limit.
- Fixed an issue with the Query Workbench that sometimes caused a Java heap space error when a query returned a large number of documents (over 1 million).

Fusion

- Fixed a bug in the Rules Editor that prevented the pagination from working as expected.

Connectors

- Fixed a bug that sometimes prevented V2 connectors that use multiple phases to fetch documents in later phases.

Predictive Merchandiser

- A bug is fixed that resulted in different filter behaviors between the API and the UI. Filters now apply to all zones.
- Fixed a bug that occasionally caused all rules to show as pinned rules if a multiple zones shared a pinned rule.
- Fixed a bug that prevented rules with disallowed symbols from showing when the **Show Applied Rules** button was clicked in the **Details** view. Symbols are now encoded.
- Fixed a bug that prevented product groups from opening when facets were selected.
- Fixed an issue with Predictive Merchandiser templates that resulted in zones and templates remaining available after the associated app was deleted and recreated.

Other changes

- The Evaluate QnA Pipeline job is now called *Smart Answers Evaluate Pipeline*.
- The **sql-jdbc service**, which serves the stateful JDBC connections, is now restricted to a single pod.

Notable documentation updates

Connectors download and installation

The connectors documentation is updated to address user feedback and clarify how connectors are downloaded and installed in each version of Fusion. The changes include:

- New how-to pages:
  - *Install A Connector - Fusion 4.x*
  - *Install A Connector - Fusion 5.1*
  - *Install A Connector - 5.2 and later*
  - *Update A Connector*
  - *Delete A Connector*
- A link is added to the Connectors index page that includes to the correct steps to install a connector, depending one the version of documentation you are viewing the *Connectors index page*.
- Corrections are made so the connector documentation accurately includes related topics, depending on the version of documentation you are viewing.

Business Rules

The Business Rules documentation is totally revamped for all versions of the Fusion docs. The changes include:

- New concepts pages:
  - *Overview*
  - *General Configurations*

    - *Conditions*
    - *Actions*
    - *Custom Rule Actions*
- New reference pages:
    - *General Configurations*
    - *Conditions*
    - *Actions*
    - *Custom Rule Actions*
- A new how-to page:
    - *Use the Query Elevation Component*
- A rewritten how-to page with clearer and more accurate instructions:
    - *Create Custom Rule Actions with the API*

API reference docs

The REST API reference docs are updated to be more accurate and make topics easier to discover. Changes include:

- A reorganized index page that makes API topics easier to find.
- The sidenav is reorganized to reflect the changes.
- Descriptions are added and updated for each API.

To get started reading the improved API documentation, see *REST API Reference*.

Deprecations

- *Sharepoint Online V1 connector*

This connector has been deprecated as of 5.3.0. Alternatively, Lucidworks recommends using the *V2 connector*.

- The *JDBC V1* connector is deprecated in 5.3 and will be removed in 5.4. In Fusion releases 5.4 and later, use the *JDBC V2 connector*.
- The config-sync Fusion microservice was deprecated and removed. It is no longer available in any release of Fusion, including Fusion 5.3.

Removals

- The AWS S3 V1 connector is not available after Fusion 5.3.
- The **System > Solr Clusters** panel is removed from the Fusion UI.

Known issues

Fusion

- Database credentials can be visible in the Indexing Service Logs. *This issue* is fixed in Fusion 5.4.
- Some log messages are dropped when encoding them as JSON in the Logging index pipeline stage. This issue is fixed in Fusion 5.4.
- In the Jobs configuration menu, the **weight field** isn't passed into the pull data, causing an error. This issue is fixed in Fusion 5.4.
- Field mappings are not retained in web datasources. This issue is fixed in *Fusion 5.3.3* and 5.4.
- Datasources API call responses contain stack traces. This issue is fixed in *Fusion 5.3.3* and 5.4.
- Evaluation jobs fail to push data to Solr when non-strings are used in the **Return Fields** field. This issue is fixed in Fusion 5.3.1.
- In the **Query Workbench**, highlighted results items are displayed in Solr syntax instead of Domain Specific Language (DSL) syntax. *This issue* is fixed in Fusion 5.4.
- Additional fields in search results disappear when many items are pinned in the **Indexing Workbench**. *This issue* is fixed in Fusion 5.4.

Predictive Merchandiser

- Basic security roles that are set up in Fusion do not work as expected with Predictive Merchandiser's *Templating API* permission. This issue is Fixed in Fusion 5.4.
- Creating a template that does not contain a main results list makes the close button unusable. This issue is fixed in Fusion 5.4.
- Using the dialog to insert a pin causes the page to refresh and switch templates. This issue is fixed in Fusion 5.4.
- Pinned results do not stay in the intended position. *This issue* is fixed in Fusion 5.4.

Connectors

- The delete by ID feature in the Access Control Collection (ACL) can potentially delete the wrong documents if the ID includes whitespace or special characters. *This issue* is fixed in Fusion 5.4.
- When adding gRPC authentication headers, the *Machine Learning Index Stage* fails to retrieve the ServiceAccount JWT value while checking `SecurityContextHolder.getContext().getAuthentication()` if ServiceAccount is set in thread context. This results in authentication errors when processing messages coming from Pulsar Subscriptions.
- The Sitecore connector fetches the wrong `ItemPath`. This issue is fixed in Fusion 5.4.
- The Active Directory (AD) V2 connector displays an `Error when processing page LdapUserPage` message while crawling. *This issue* is fixed in Fusion 5.4.
- The Windows Share SMB 2/3 V2 connector displays a `STATUS_NOT_SUPPORTED` error. This issue is fixed in Fusion 5.4.
- Validation requests in the Sitecore connector plugin time out due to a `classpath` issue. This issue is fixed in Fusion 5.4.
- The gRPC service (`connectors-rpc`) does not start when a Fusion deployment is behind a proxy. *This issue* is fixed in Fusion 5.4.
- The V2 connector runtime quickly runs out of memory. *This issue* is fixed in Fusion 5.4.
- The JIRA Connector displays an error when signatures are null. *This issue* is fixed in Fusion 5.4.

Fusion 5.2.2

Release date: October 20, 2020

Component versions:

Component	Version
Solr	8.4.1
ZooKeeper	3.5.7
Spark	2.4.5
Kubernetes	GKE, AKS, EKS 1.18 Rancher (RKE) and OpenShift 4 compatible with Kubernetes 1.18 OpenStack and customized Kubernetes installs not supported. See *Kubernetes support* for end of support dates.
Ingress Controllers	Nginx, Ambassador (Envoy), GKE Ingress Controller Istio not supported.

More information about support dates can be found at *Lucidworks Fusion Product Lifecycle*.

	Looking to upgrade? Check out the *Fusion 5 Upgrades topic* for details.

	This patch addresses an issue that can cause a serious performance issue for high-scale deployments. Contact your Customer Excellence representative if you have questions about the potential impact to your deployment.

Bug fixes

Fusion

- A serious bug is fixed that, under specific conditions, resulted in Fusion sending requests to the same replica for a specific Solr shard for every request. This effectively disabled load balancing, causing significant performance issues.

Known Issues

Fusion

- The Partial Update Indexer stage sets multivalued fields to an empty string `""` instead of deleting them. *This issue* is fixed in Fusion 5.4.
- A conflict between `jsr311-api.jar` and `javax.ws.rs-api` causes SharePoint to be unable to index content files. This issue is fixed in Fusion 5.2.3, 5.3.1, and 5.4.

Connectors

- The pipeline stages skip documents, causing the RPC service to display an error. This issue is fixed in Fusion 5.3.3 and 5.4.

Fusion 5.2.1

Release date: September 23, 2020

Component versions:

Component	Version
Solr	8.4.1
ZooKeeper	3.5.7
Spark	2.4.5
Kubernetes	GKE, AKS, EKS 1.18 Rancher (RKE) and OpenShift 4 compatible with Kubernetes 1.18 OpenStack and customized Kubernetes installs not supported. See *Kubernetes support* for end of support dates.
Ingress Controllers	Nginx, Ambassador (Envoy), GKE Ingress Controller Istio not supported.

More information about support dates can be found at *Lucidworks Fusion Product Lifecycle*.

Looking to upgrade?
Check out the *Fusion 5 Upgrades topic* for details.

A patch is released for this version that addresses an issue that can cause a serious performance issue for high-scale deployments. Please upgrade from Fusion 5.2.1 to 5.2.2+ as soon as possible. Contact your Customer Excellence representative if you have questions about the potential impact to your deployment.

Improvements

Connectors

- Replaced P12 key with JSON for Google Drive Classic service account.
- The Connectors API returns the intermediate Deploying and Deleting states to improve the installation and deletion experience. This allows the UI to correctly categorize the displayed connectors list with appropriate state-tags, colors, and labels.
- V2 connectors index content via `key_shared` indexing subscriptions. This helps increase indexing performance while maintaining ordering of same document ID updates to Solr.
- FS connectors have an additional configuration option with Attitiona mounting points to use the FS connector in a cloud environment.

Fusion

- The Query Rewrite ML jobs have a frequency column generated that is shown in the UI. This column should be used to sort in descending order and focus on the high frequency rewrites.

Bug fixes

Fusion

- In Fusion 5.2, there is no `/api/apollo/*` API, so the Banana default API settings use `/api/*` instead.
- Fixed a bug where foreign characters were skipped during query rewrites.

Connectors

- Prometheus metrics are integrated with connectors REST, classic, and connector-plugin services.
- Creating a new datasource fills performance setting parameters with default values for classic connectors.
- Fixed a bug where config validation failed when provided credentials had S3 bucket access permission but not bucket ownership.
- Installing from the connector repository installs the latest plugin available.
- Connectors can now decrypt datasource config after a migration from 5.1.4 to 5.2.0.

Predictive Merchandiser

- The "Show applied rules" modal works in Details view even when a document is missing signals due to symbol encoding errors.
- Fixed a bug where facets were not consistently applied to zones.
- Facets shown in URLs are reapplied on page reload even when cookies are reset.
- When a user navigates away from unsaved template and zone edits, a confirmation modal is shown.
- The initial default Home template is renamed as Search Results.

Other changes

- The most recent Connectors SDK version can be found at https://github.com/lucidworks/connectors-sdk/releases/tag/v2.0.3 *(https://github.com/lucidworks/connectors-sdk/releases/tag/v2.0.3)*.
- Before starting a crawl, FTP Pytests checks for a subdirectory and creates the folder if it does not exist.
- Updated Banana version to **banana-fusion@1.6.27**.

Known Issues

Connectors

- The Google Cloud Storage connector displays an API error after upgrading from 5.1.2 to 5.2.2. This issue is fixed in Fusion 5.3 and 5.4.
- Datasources for Google Drive, SharePoint Online, and Confluence connectors cannot be deleted. This issue is fixed in Fusion 5.3.

Fusion

- The request between replicas is not balanced correctly when `shards.preference` is set on a request in SolrJ, under certain conditions. This issue is fixed in Fusion 5.1.5 and 5.2.2.
- Text and Query Analysis jobs are not saved when added with the **Add+** button. This issue is fixed in Fusion 5.3.

Predictive Merchandiser

- The software does not use configured fields to display thumbnails, titles, and descriptions. This issue is fixed in Fusion 5.3.
- The Admin UI shows garbled characters when files that contain Japanese characters are opened for edit. This issue is fixed in Fusion 5.3.
- Rules show in Unicode and not the user-designated language. This issue is fixed in Fusion 5.3.
- The Admin UI does not behave as intended when changing the directional configuration of a rule. This issue is fixed in Fusion 5.3.

- The Synonym Editor in Query Rewrite does not behave as intended when typing Japanese characters in the field. This issue is fixed in Fusion 5.3.
- Pinned results do not stay in the intended position. *This issue* is fixed in Fusion 5.4.

Pulsar

- `nettyMaxFrameSizeBytes` specification is not set for Bookkeeper, causing clusters to run out of storage when a message is larger than 5 MB. This issue is fixed in Fusion 5.3.

Fusion 5.2.0

Release date: August 24, 2020

Component versions:

Component	Version
Solr	8.4.1
ZooKeeper	3.5.7
Spark	2.4.5
Kubernetes	GKE, AKS, EKS 1.18 Rancher (RKE) and OpenShift 4 compatible with Kubernetes 1.18 OpenStack and customized Kubernetes installs not supported. See *Kubernetes support* for end of support dates.
Ingress Controllers	Nginx, Ambassador (Envoy), GKE Ingress Controller Istio not supported.

More information about support dates can be found at *Lucidworks Fusion Product Lifecycle*.

	Looking to upgrade? Check out the *Fusion 5 Upgrades topic* for details.

New Features

Fusion

Fusion DSL

	The beta release of Fusion Domain Specific Language (DSL) functions are available beginning with Fusion 5.2.2.

Fusion Domain Specific Language (DSL) provides expected search results as a JSON response in a way that reduces search query complexity for the user.

Previously, users needed to understand complex syntax to express certain search queries in Fusion (for example, the best way to express a facet filter). DSL gives Fusion control over how to execute a query by transforming a structured query input into a Solr request, where we can add intelligence around the index and the user's intent.

Why Search DSL

Fusion Search Domain Specific Language (DSL) reduces search query complexity for the user. Fusion Search DSL supports expressive search queries and responses via a structured, modern JSON format.

Search DSL is an alternative to the current (now referred to as) "Legacy" format for performing search queries. Previously, users needed to understand complex syntax to express certain search queries in Fusion (for example, the best way to express a facet filter).

Compared to the legacy Solr parameter format, Search DSL is structured to more closely align with the central concepts of Fusion and provide a more usable alternative for expressing complex Fusion queries.

Search DSL gives Fusion control over how to execute a query by transforming a structured query input into a Solr request, at which time Fusion can add intelligence around the

index and the user's intent.

Limitations

Despite the advantages, there are still some important limitations to be aware of when deciding whether to use the Fusion Search DSL.

> These limitations only apply to DSL queries. Legacy queries can still be issued separately from DSL queries without being subject to these limitations.

1. **Head / Tail Rewrite.** Support is currently limited when using DSL queries:
   a. In order to work on DSL queries, the `Improved Query` must be entered in as a JSON string representing the desired `main` query that should be issued (this replaces the `queryDefinition.main` field in the rewritten DSL query)
   b. The query rewrites produced by the built-in head/tail rewrite job will NOT work on DSL queries, as the job only outputs legacy-style rewrites
2. **Query Stage Support.** All query stages are fully supported
3. **Rules Support.** All kinds of rules are fully supported with the exceptions noted below
   a. **Set Params.** Not supported
   b. **Custom Rule.** Not supported

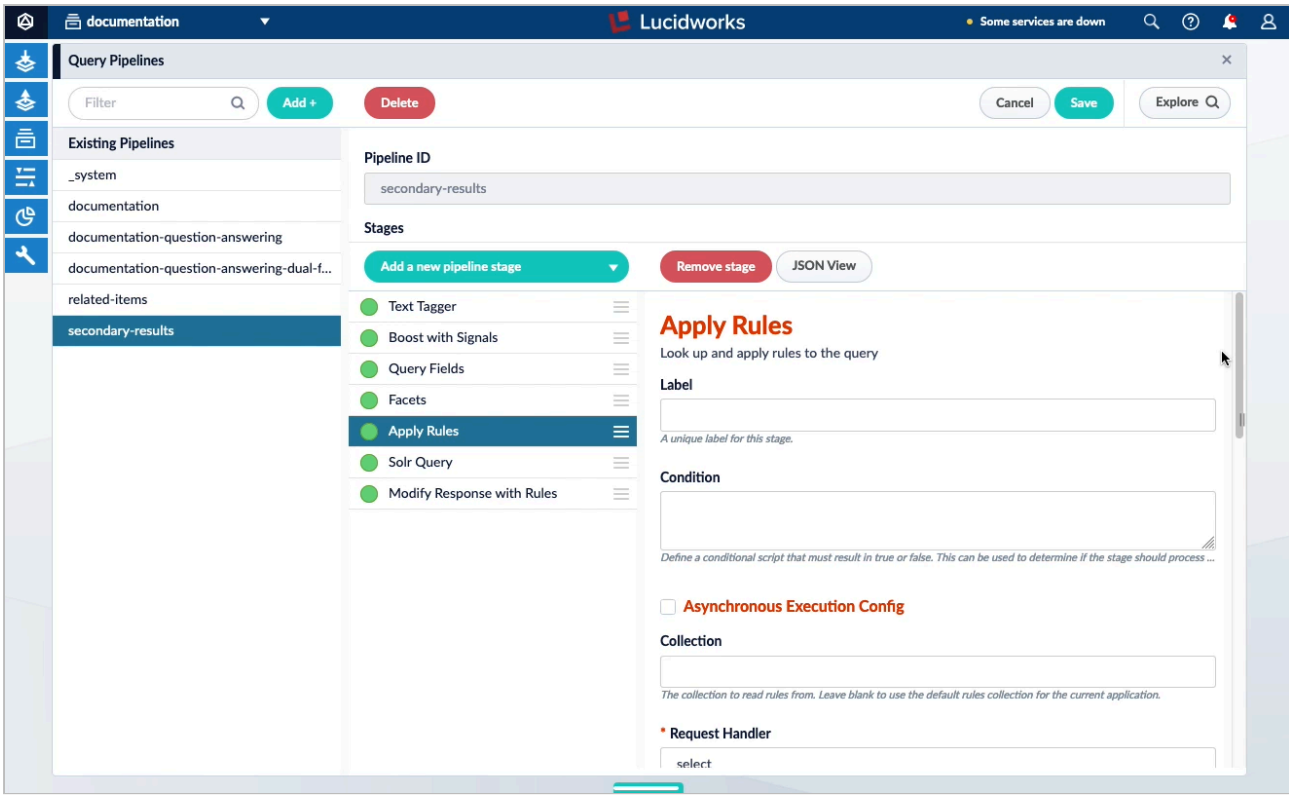
See *Domain Specific Language* for additional information, including examples.

Pipeline Stage and Blob Editor

Fusion 5.2.0 introduces the ability to create and copy pipeline stages by pasting JSON objects in the Fusion UI. Additionally, it allows users to edit certain blobs in a similar fashion. Only JSON is supported, and JSON validation is included to prevent the user from saving an invalid object.

Pipelines stages
You can copy and paste pipeline stage JSON data directly from the Fusion UI by clicking the **JSON View** button in the stage's configuration panel.

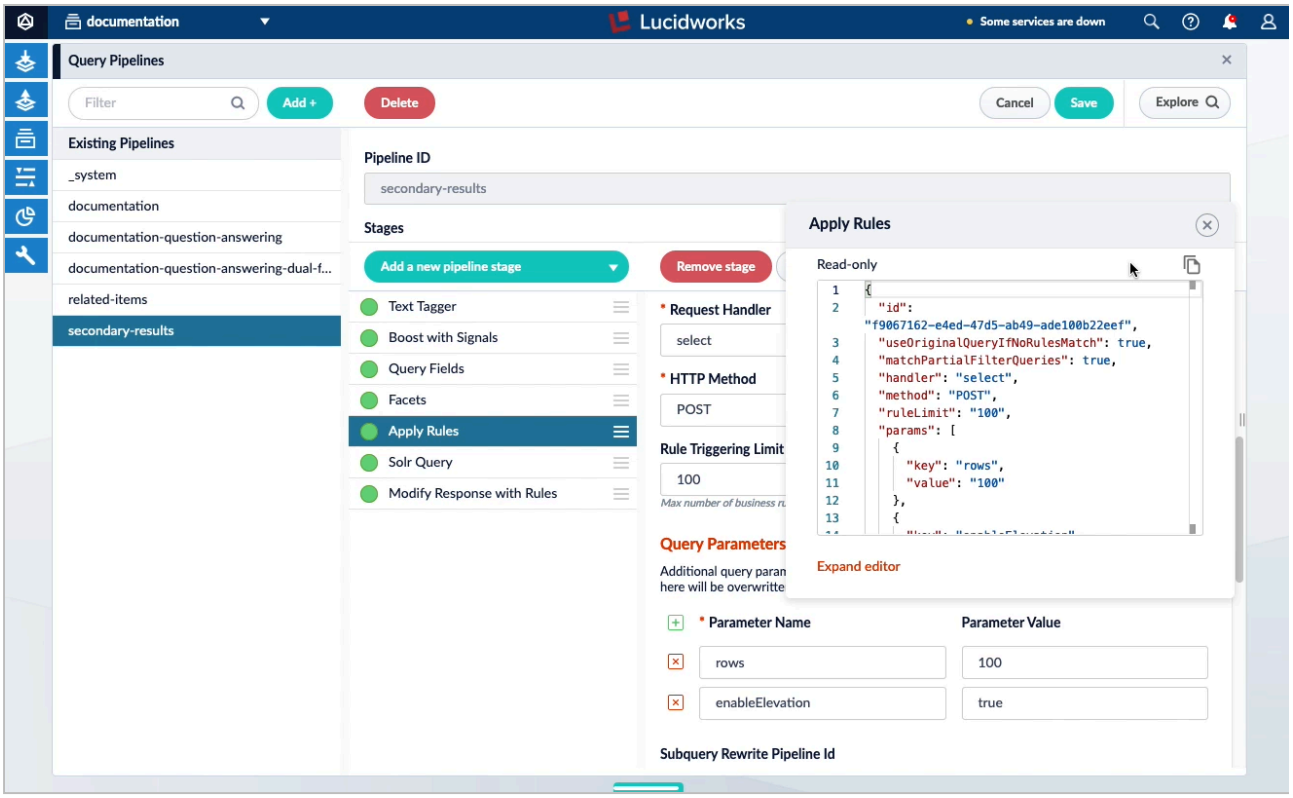
Existing stages are **READ-ONLY** and have the option to **Copy** only.



New stages will have **Copy** and **Paste** buttons. Changes are applied to the stage with the **Apply** button.

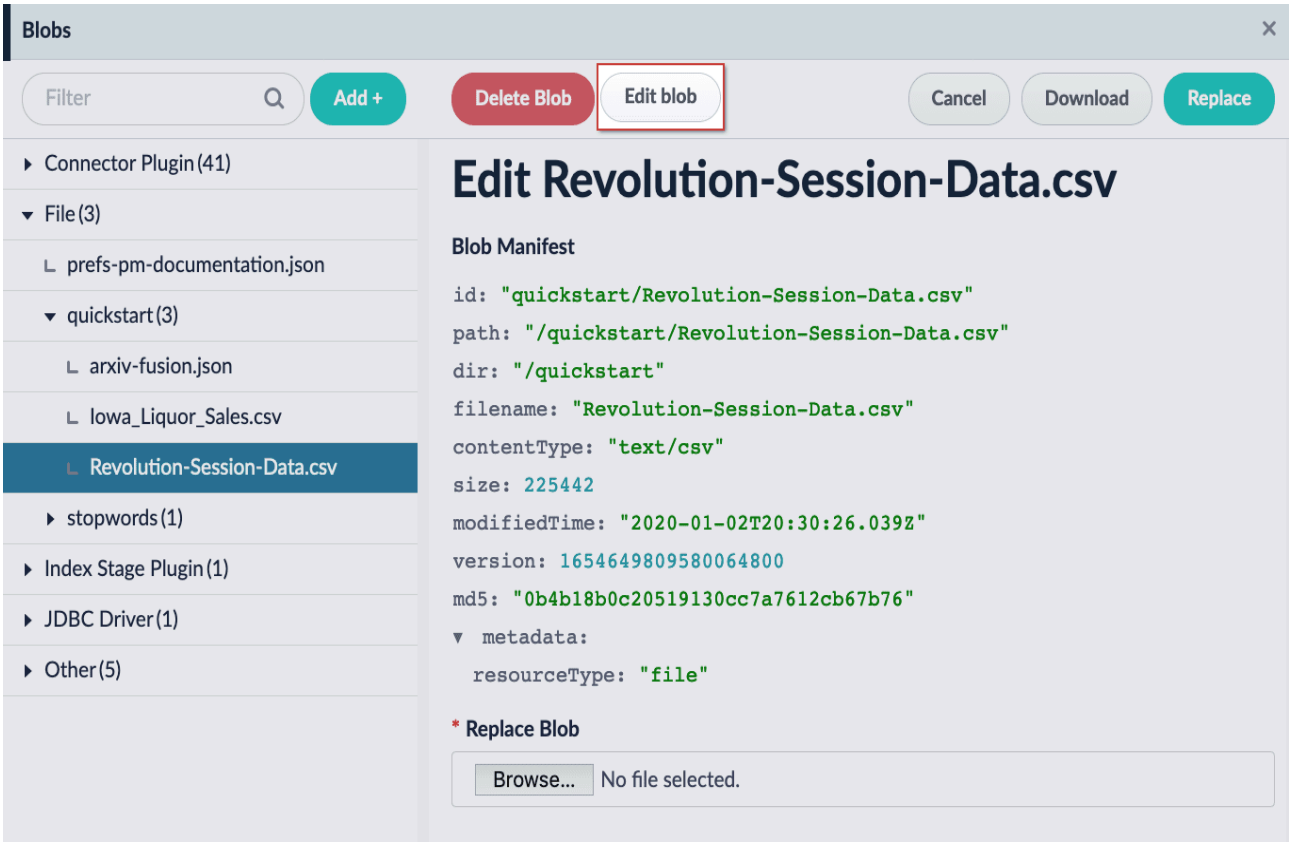
> Users will still need to **Save** the stage for the changes made in the editor to be saved.

This allows you to quickly and easily duplicate and edit stages within your app or share your configurations with others.



See *Index Pipeline Stages* and *Query Pipeline Stages* for more information. For instructions, see *Use the Pipeline Stage JSON Editor*.

Blobs
The blobs editor is accessed by clicking the **Edit blob** button in the blobs configuration panel:



The blobs editor is similar to the pipeline stage editor with the following exceptions:

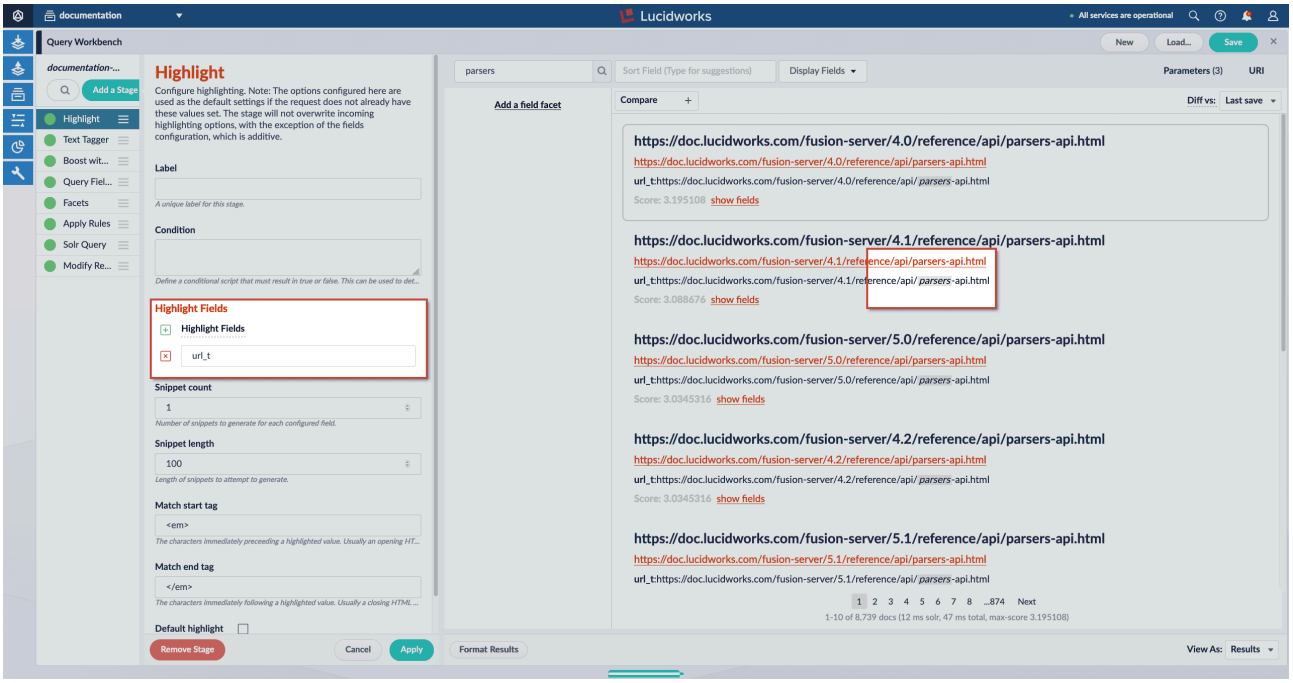
- Multiple file types are supported, including:
  - CSV
  - Java
  - JavaScript
  - Python
  - SCALA
  - Typescript
  - Plain text
- Existing blobs are *not* considered **READ-ONLY**, giving users the option to edit existing blobs.
- When the user clicks the **Save** button, changes to the blob data take effect immediately.

See *Blob Storage* for more information. For instructions, see *Use the Blob Editor*.

Highlight Query Pipeline Stage

Fusion 5.2.0 introduces a new query pipeline stage, *Highlight*.

The Highlight stage allows you to highlight key parts of fields by generating snippets and appending match tags to the highlighted terms. The highlight match tags can then be styled in your web search application.



To learn more, including how this stage differs from other highlighting options, see the *Highlight Stage* reference article. To learn how to use the Highlight stage, see *Configure the Highlight Query Pipeline Stage*.

Fusion

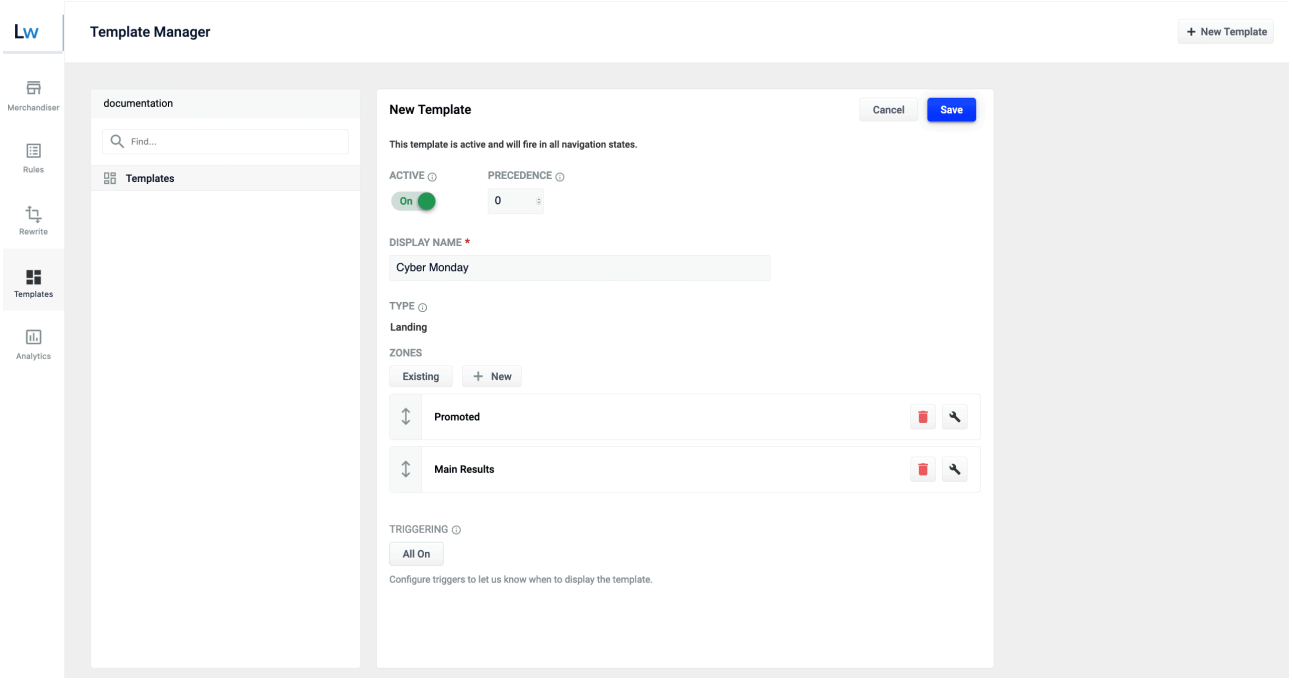
- The new *Classification job* analyzes how existing documents are categorized and produces a model that can be used to predict the categories of new documents at index time.
- The new *BPR Recommender job* produces better signals-based recommendation data with a shorter running time than the ALS Recommender job (deprecated in this release).
- The new *Content-Based Recommender job* analyzes similarities in your content to provide recommendations when you do not have enough signals data for signals-based recommendation methods.
- The new *Query-to-Query Session-Based Similarity* job provides improved query-to-query recommendations using a two-pronged approach.

- A new *Recommend Similar Queries* query pipeline stage was added to support the new query-to-query recommendations job.

Predictive Merchandiser

Templates

The Templates screen in the Rules Editor allows you to design, test, and implement a wide variety of search experiences. Results are conditionally delivered from multiple different query pipelines, enabling you to use Fusion's search, browse, and AI-driven functionality throughout your site.

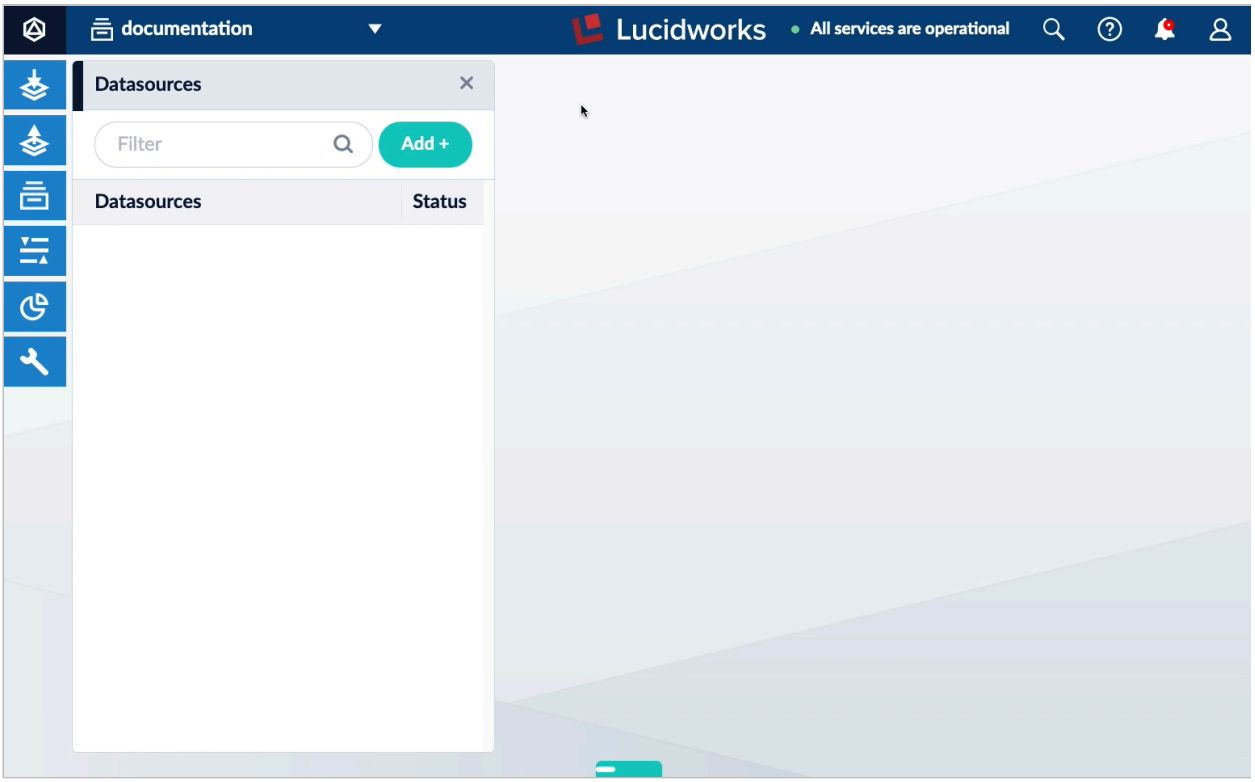


See *Predictive Merchandiser Templates* for more information. For instructions on how to use Templates, see *Use Predictive Merchandiser Templates*.

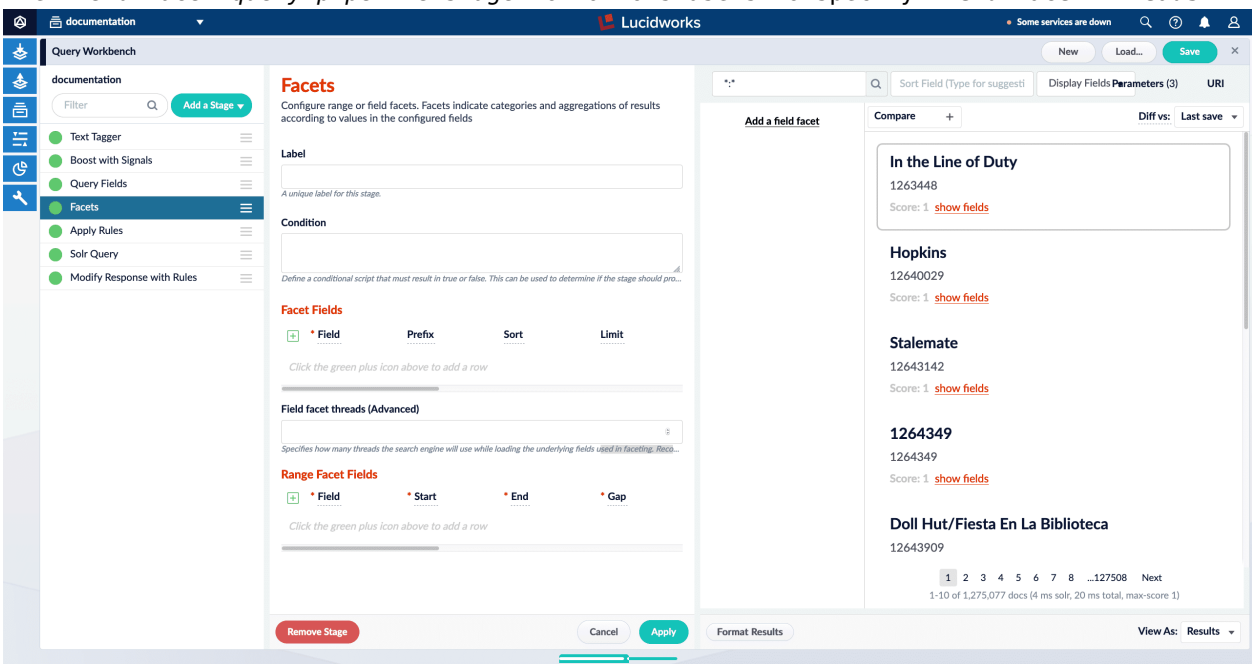
Improvements

Fusion

- Fusion SQL adds search and math expressions to the SQL experience, with new, intuitive, easy-to-learn functions that simplify the way power users query Fusion to explore unstructured text and patterns in data. In this release you can combine structured and unstructured data stored in Solr to understand numeric distributions and correlations, fit models, explore clusters, perform powerful sampling, use text analyzers for entity extraction, find significant terms, and perform time series analysis.
  Learn more about *Fusion SQL* or explore the new functions for these features:
  - ***search***
  - ***text analytics***
  - ***aggregation***
  - ***time series (date_format)***
  - ***statistics***
  - ***miscellaneous***
- You can now download connectors directly from the Fusion UI. Powered by the *Connector Plugins Repository API*, you can easily view which connectors are available to you and download new connectors with a single click. Navigate to **Indexing > Datasources**, click **Add**, and click the name of the connector you want to install:



- The *Field Facet query pipeline stage* now allows users to specify **Field facet threads:**



> Specifies how many threads the search engine will use while loading the underlying fields used in faceting. Recommended that this is kept at `0` or blank, which means the data is loaded in the thread already running the query.

- Various Spark jobs improvements:
  - Improved logging
  - All Spark writing operations now support `min-output-partitions`
  - *SQL aggregation Jobs* are audited for natural key usage
  - Recommender aggregation
- Previously, the *Apache Pulsar* client automatically blocked message objects if the queue was full. Now, this behavior is configurable as a boolean value. By default, this is

set to `false` in the query service to ensure errors are successfully logged even if the Pulsar service becomes available. Outside of the query service, the default is `true`.

- Pulsar is updated to version 2.6.0.
- The Query Workbench no longer displays additional pages if there are not enough results to populate those pages.
- The Query Workbench now displays error messages instead of writing error messages to the console only.
- `AsyncJoinQueryStage` is updated to always produce an error message to improve debugging.
- You can now configure your realm with additional headers to check for allowed IPs. This is useful when your gateway is behind a proxy which is replacing the incoming IP.
- A new Index Stage SDK API method is added: `{{Document.allFields()}}`. This method returns all document fields, including internal fields. (Internal fields start with the `{{lw}}` prefix.)

Fusion

- Solr's `zkhost` is now included in the *Jupyter* environment. It no longer needs to be explicitly defined for each Jupyter notebook.
- You can now use more than 30 parameters in a single business rule.

Predictive Merchandiser

- Publishing performance is significantly improved.

Other changes

Fusion

- Pipeline context can now properly serialize/deserialize arbitrary values and nested maps/lists when sent via Pulsar Message.
- The Pulsar default namespace is now set when creating or updating an index pipeline or profile. Previously, this was set when the stage was first enacted.
- All paths starting with `/apollo` are removed.
- Miscellaneous UI improvements.
- The Rules Editor's Dashboard is renamed to Analytics.

Predictive Merchandiser

- Enabled the ability to use nonexistent fields and values when creating a rule.
- Miscellaneous UI improvements.

Bug fixes

Fusion

- Fixed a bug which sometimes prevented indexing of access control items at the end of a connector job.
- The *AD ACL V2 connector* now uses the `group.members` field instead of `user.memberOf`. This addresses a change in Windows Server 2019 that caused incremental crawls to fail.
- Fixed a bug that prevented *Prometheus metrics* from generating for connectors.
- Fixed a bug that sometimes prevented job status icons from updating.
- Fixed a bug with the job service that sometimes resulted in certain error conditions to create multiple, redundant method calls.
- Fixed a bug that prevented Spark executors from receiving the Prometheus pushgateway address.

Fusion

- Fixed a bug that sometimes caused the *Document Clustering job* to fail if a `trainingDataFilterQuery` was used.

Predictive Merchandiser

- Fixed a bug that prevented *item-for-item recommendations* from displaying in the **Details** panel.
- Fixed a bug that prevented facet names from appearing in Predictive Merchandiser if the facets were created in a query pipeline using the *Field Facet stage*.
- Fixed a bug that caused errors in the Predictive Merchandiser UI when switching between query profiles with grouping in use and query profiles without.
- Fixed a bug that prevented a `set-params` rule from saving successfully multiple parameters are defined.

Deprecations

- The config-sync Fusion microservice was deprecated and removed. It is no longer available in any release of Fusion, including Fusion 5.2.

> The following features and functionalities are deprecated in Fusion 5.2.0 and will be removed in a future release.

- *Time-based partitioning*
- The *Blob Store's* searchable aspects
- MLeap in Machine Learning models
- MLeap deployments of SpaCy and SparkNLP in favor of *Seldon Core deployments*
  > Your Seldon Core deployments will continue to work in Fusion 5.2.0+.
- Various Fusion jobs:
  - *Query-to-Query Collaborative Similarity Job*
    > We recommend the use of the *Query-to-Query Session-Based Similarity jobs*, which provides better performance and query coverage.
  - *ALS Recommender Job*
    > We recommend the use of the experimental *BPR Recommender job*, which provides better results with a shorter running time. The ALS Recommender job will be removed completely when the BPR Recommender job goes General Availability (GA).
  - *Logistic Regression Classifier Training Jobs* and *Random Forest Classifier Training Jobs*
    > We recommend the use of the new *Classification job*, which provides more options and better logging.
  - *Word2Vec Model Training Jobs*
- Various Fusion pipeline stages:
  > These stages are deprecated in favor of new forthcoming Seldon Core functionality. These stages will be removed when the replacements are in General Availability (GA).
  - *NLP Annotator Index Stage*
  - *OpenNLP NER Extraction Index Stage*
  - *NLP Annotator Query Stage*

Removals

> The following features and functionalities are removed in Fusion 5.2.0.

- Various Fusion jobs:
  - Co-occurrence Similarity Jobs
  - Collection Analysis Jobs
  - Item Similarity Recommender Jobs
  - Legacy Item Recommender Jobs
  - Legacy Item Similarity Jobs
  - Levenshtein Spell Checking Jobs
  - Matrix Decomposition-Based Query-Query Similarity Jobs
  - SQL-Based Experiment Metric Jobs
  - Statistically Interesting Phrases Jobs

Known Issues

Connectors

- Requests with the headless browser in the user agent are rejected. This issue is fixed in Fusion 5.4.2.
- The SharePoint connector does not apply lowercasing to incoming records. This issue is fixed in Fusion 5.4.

Signals

- The **Send click signals** function does not populate signals collections. This issue is fixed in Fusion 5.2.1 and 5.3.

- Training on the BPR recommendation job when metadata settings are specified will fail.
  Workaround Steps:
  1. Download the *shell script* from here.
  2. Run the script in a terminal `./rec-5.2.0-workflow-fix.sh <namespace>`

	Replace `<namespace>` with the k8s namespace where Fusion is deployed.
- Running the recommendation or classification workflows may result in pods failing with a docker image pull error.
  Workaround Steps:
  1. Download the *shell script*.
  2. Run the scrupt in a terminal `./workflow-5.2.0-tag-fix.sh <namespace>`

	Replace `<namespace>` with the k8s namespace where Fusion is deployed.

Fusion 5.1.5

Release date: October 20, 2020

Component versions:

Component	Version
Solr	8.4.1
ZooKeeper	3.5.6
Spark	2.4.5
Kubernetes	GKE, AKS, EKS 1.18 Rancher (RKE) and OpenShift 4 compatible with Kubernetes 1.18 OpenStack and customized Kubernetes installs not supported. See *Kubernetes support* for end of support dates.
Ingress Controllers	Nginx, Ambassador (Envoy), GKE Ingress Controller Istio not supported.

More information about support dates can be found at *Lucidworks Fusion Product Lifecycle*.

	Looking to upgrade? Check out the *Fusion 5 Upgrades topic* for details.

	This patch addresses an issue that can cause a serious performance issue for high-scale deployments.Please upgrade from Fusion 5.1.4 to 5.1.5 as soon as possible. Contact your Customer Excellence representative if you have questions about the potential impact to your deployment.

Bug fixes

Fusion

- A serious bug is fixed that, under specific conditions, resulted in Fusion sending requests to the same replica for a specific Solr shard for every request. This effectively disabled load balancing, causing significant performance issues.

Fusion 5.1.4

Release date: June 25, 2020

Component versions:

Component	Version
Solr	8.4.1
ZooKeeper	3.5.6
Spark	2.4.5
Kubernetes	GKE, AKS, EKS 1.18 Rancher (RKE) and OpenShift 4 compatible with Kubernetes 1.18 OpenStack and customized Kubernetes installs not supported. See *Kubernetes support* for end of support dates.
Ingress Controllers	Nginx, Ambassador (Envoy), GKE Ingress Controller Istio not supported.

More information about support dates can be found at *Lucidworks Fusion Product Lifecycle*.

	Looking to upgrade? Check out the *Fusion 5 Upgrades topic* for details.

	A patch is released for this version that addresses an issue that can cause a serious performance issue for high-scale deployments. Please upgrade from Fusion 5.1.4 to 5.1.5+ as soon as possible. Contact your Customer Excellence representative if you have questions about the potential impact to your deployment.

Bug fixes

Fusion

- Fixed bug that caused Spark jobs to hang.
- Clusters upgraded to 5.1.3 had incompatible plugins in the blob store. Classic plugins installed in a previous version will be upgraded on migrations.
- More helpful error messages from Solr are now transferred to the admin UI to make it easier to debug queries.

Known Issues

Fusion

- Filters with commas in the values do not trigger rules. This issue is fixed in Fusion 5.4.
- Setting `shards.preference` can break Solrj load balancing under certain conditions. This issue is fixed in Fusion 5.1.5 and 5.2.2.
- The Partial Update Index Stage sets multivalued fields to an empty string instead of deleting them. This issue is fixed in Fusion 5.4.
- Pinned items can change positions when multiple rules are created. This issue is fixed in Fusion 5.3.
- Applying **Return Fields** to a pipeline causes the specified field to appear blank. This issue is fixed in Fusion 5.2.

- The **Enable Recommendations** feature appears in user options after initial activation, causing an API error when clicked. This issue is fixed in Fusion 5.2
- The Query service returns `Null Subject` when sending security trimming requests from Connectors. This issue is fixed in Fusion 5.2.
- Client secrets are displayed in plain text in the OpenID Connect Fusion realm configurations. This issue is fixed in Fusion 5.2.
- The **Schedule Status** checkbox is shown as a triCheckbox instead of a normal checkbox. This issue is fixed in Fusion 5.2 and 5.3.

Connectors

- The SharePoint V2 connector shows file types and documents that should be filtered out. This issue is fixed in Fusion 5.3
- Default UI field mappings for the Confluence connector cannot be disabled. This issue is fixed in Fusion 5.3.3 and 5.4.
- The Confluence connector indexes file extensions that are not included as approved extensions. This issue is fixed in Fusion 5.3.3. and 5.4.
- The Active Directory V2 Connector does not index users. This issue is fixed in Fusion 5.3.
- Classic Connectors show blank datasource configuration fields. This issue is fixed in Fusion 5.2.1 and 5.3.
- Plugin failures cause the service to go into a bad state, but no indications of errors are shown in the UI. This issue is fixed in Fusion 5.2.1 and 5.3.

Predictive Merchandiser

- Special characters cause a 400 error in the product info modal. This issue is fixed in Fusion 5.3.
- Recommendations for items are shown in the wrong order. This issue is fixed in Fusion 5.2 and 5.3.

Spark Jobs

- Custom Spark Jobs can cause the Spark session to crash. This issue is fixed in Fusion 5.4.

Fusion 5.1.3

Release date: June 18, 2020

Component versions:

Component	Version
Solr	8.4.1
ZooKeeper	3.5.6
Spark	2.4.5
Kubernetes	GKE, AKS, EKS 1.18 Rancher (RKE) and OpenShift 4 compatible with Kubernetes 1.18 OpenStack and customized Kubernetes installs not supported. See *Kubernetes support* for end of support dates.
Ingress Controllers	Nginx, Ambassador (Envoy), GKE Ingress Controller Istio not supported.

More information about support dates can be found at *Lucidworks Fusion Product Lifecycle*.

	Looking to upgrade? Check out the *Fusion 5 Upgrades topic* for details.

New Features

Fusion

- Fusion 5.1.3 adds the *Connector Plugins Repository API*. This API is used to install and update connector plugins from the built-in plugin repository. For example, the following command describes a successful installation of a connector plugin with the API:

```
curl -u USERNAME:PASSWORD -X POST
"http://FUSION_HOST:FUSION_PORT/connectors/plugins?id=lucidworks.fs" {
"pluginId" : "lucidworks.fs", "status" : { "action" : "INSTALL", "status" :
"SUCCESS", "message" : "Plugin lucidworks.fs was installed successfully." } }
```

Improvements

Fusion

- Support for `service://` and `extservice://` is added to the *REST Query Index Stage*. This is consistent with the *Rest Call Jobs*, which supports calls to other Kubernetes services. For example, `service://SERVICE_NAME/<path>` or `extservice://SERVICE_NAME/<path>`.
- The *Solr Subquery* query pipeline stage now supports grouped results from Solr grouped queries. Access the `groupValue` and the documents for a group in the Javascript stage:

```
groupValue = ctx.get("subquery-
results").grouped.get("Category_s").getGroups().get(0).getGroupValue() docs =
ctx.get("subquery-
results").grouped.get("Category_s").getGroups().get(0).getDoclist().docs
```

Now, Solr Subquery's `SearchResponse` has a new field, `grouped`, which is a map of group field names to `GroupCollections`.

Other changes

Fusion

- The Push connector is removed. As an alternative, directly access the index endpoints.
- Java-based services have moved to distroless Docker images. As a result, debugging tools must be run remotely.
- The default *connectors* included with Fusion have changed. See the table below for the connectors sorted by *platform version*:

V1	V2
Web V1	*SharePoint*
JDBC V1	*Active Directory Connector for ACLs*
File Upload	*Local Filesystem*
Confluence	*OneDrive*
JIRA	*Sitecore*
	Couchbase

Bug fixes

Fusion

- Fixed a bug that prevented the *FTP connector* from working.
- Fixed a bug that caused long data values in job statuses to exceed the allowed character limit.

Known Issues

Fusion

- Empty operations can be saved instead of failing. *This issue* is fixed in Fusion 5.3.
- The **Schedule Status** checkbox is shown as a triCheckbox instead of a normal checkbox. This issue is fixed in Fusion 5.2 and 5.3.
- Spark jobs fail and display a `NoSuchMethodError`. This issue is fixed in Fusion 5.1.4 and 5.2.
- Content-based recommender jobs fail because they are unable to find the config map. This issue is fixed in Fusion 5.2.
- Imported and exported apps do not include subscriptions. This issue is fixed in Fusion 5.2.

Fusion 5.1.2

Release date: May 14, 2020

Component versions:

Component	Version
Solr	8.4.1
ZooKeeper	3.5.6
Spark	2.4.3
Kubernetes	GKE, AKS, EKS 1.18 Rancher (RKE) and OpenShift 4 compatible with Kubernetes 1.18 OpenStack and customized Kubernetes installs not supported. See *Kubernetes support* for end of support dates.
Ingress Controllers	Nginx, Ambassador (Envoy), GKE Ingress Controller Istio not supported.

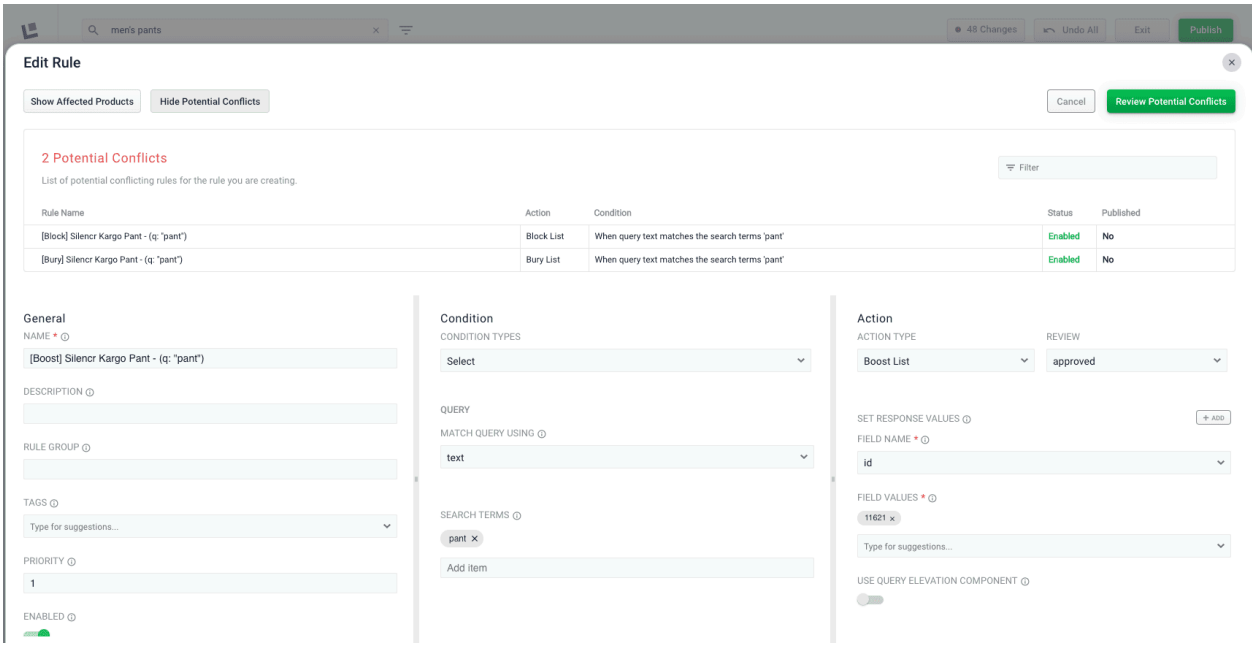
More information about support dates can be found at *Lucidworks Fusion Product Lifecycle*.

	Looking to upgrade? Check out the *Fusion 5 Upgrades topic* for details.

New Features

Predictive Merchandiser

- Rules can now be created for product groups in Predictive Merchandiser. Product groups can be boosted or blocked. If you want to promote a product group that's on sale, for example, you can boost the group to the top of the results. You can still apply rules individually to products within a group. This allows you to boost a product based on a specific user's search behavior. If a user typically favors the color red, you can boost the red variant of the product to the top of the group.
  Read more about *product grouping* in Predictive Merchandiser.
- Predictive Merchandiser allows search engineers to ensure users do not encounter a *zero results page* by creating rules to display select products when a zero results page is reached. Read more about how to *resolve zero result queries* with Predictive Merchandiser.
- The UI now shows potentially conflicting or similar business rules when creating or editing rules. For example, if you have 3 rules that boost, bury, and block the same product, Predictive Merchandiser highlights these conflicting rules so you can address them.



Read more about the feature in the *Rule conflict resolution*.

Improvements

Fusion

- The Query Rewriting API adds a new endpoint, `/targeted-docs`, that lists which rules affect a particular document.
- Custom Index Stages now let user know they cannot be deleted if they are in the blob store and being used by the Fusion index pipeline.
- Added row control and Enabled/Disabled column toggle functionality to Affected Rules table.
- Tail rewrites based on reasons of 'number', 'spelling', 'rare-term', or 'others' are removed from the staging collection, but are retained for the reports collection.
- Kubernetes has a 63-character limit on pod names due to DNS naming. Spark job names are now truncated to comply with Kubernetes limits during pod creation.
- You can now add annotations to Solr pods on a per-node pool basis by setting annotations on the Solr `statefulset`.
- Rule tables rows now have a conflict indicator that pops out on hover notifying the user there is another rule that potentially conflicts with this one.
- Performance for signals boosting via boost parameter has been improved.
- Unpublished rules list loads 25 items upon opening and fetches another 25 from API when user scrolls to the end of the list to imitate infinite scroll.
- Write option for configuring `commit_within` added to the ALS recommender config to prevent overwhelming Solr when running a large recommendation job. We suggest setting this to `{{commit_within=-1}}`.

Fusion

- The Synonym Rules UI previously showed synonym pairs by default. The job now creates new grouped rules based on whether the job determines pair or context. If the job predicts `context`, then synonym pairs will display as context pairs.
- Synonym context will be shown as default if the job predicts context. UI will allow users to switch between suggestions per synonym.

Predictive Merchandiser

- Hovering over rules in the list now shows rules affected.

- Users can now see a list of applied rules for a product to help determine which rules are affected when creating a new rule or editing an existing one.

Other changes

Fusion

- The Rewrites dashboard defaults have been updated to remove misspelling and head/tail columns and hide phrase and synonym columns.
- When classic/rpc are started for the first time, the plugins from bootstrap path are installed by default. Now it is possible to decide whether to install the plugins from the bootstrap. Before deploying classic or rpc, modify via configMap: from `bootstrapEnabled: true` to `bootstrapEnabled: false`.
- The helm chart variable `solrAutocalingPolicyJson` for `fusion-admin` has been deprecated in favor of `solrAutoscalingPolicyJson` and will be removed in a future release. Please update any custom values files that reference this variable.
- The Pulsar client option to block the message queue if full is now configurable. By default, this boolean is set to true. In the query service this value is set to false.
- Messaging Stage Configs are now deprecated.
- Upgraded Jackson to 2.9.10 and Jackson bind to 2.9.10.4.
- Upgraded Fusion to use Tika 1.24.

Fusion

- Synonym detection confidence scores are rescaled so the final output appears more significant than the default threshold.
- Misspelling confidence scores are changed from numerical values of 0, 0.5, and 1 to categorical values of low, medium, and high, respectively.

Predictive Merchandiser

- Licensing for Predictive Merchandiser is updated. With a valid license, users have access to the *Dashboard*, *Merchandiser*, *Rules*, and *Rewrites* screens in the Rules Editor. Without a license, users have access to the Rules, Rewrites, and Simulator screens.

  > The Rules Editor requires an *Advanced* tier license. See Pricing Plans *(https://lucidworks.com/pricing-plans)* for more information.

- details can now truncate values for multiple purchases in the bar display.
- Fusion now has two additional rule types, `in_group_boost` and `in_group_bury`, which are designed to boost and bury documents within a particular "group" in Solr requests that use standard grouping *(https://lucene.apache.org/solr/guide/8_5/result-grouping.html)*. These rule types should be chosen over the traditional "Boost List" and "Bury List" rules when users want to reorder documents within a group without affecting the ordering of the groups themselves.
- Users can now see a list of affected rules for each product.
- There is now an additional active date facet.
- Conflict resolution added inside rules manager edit modal to enable/disable conflicting rules.

Connectors

- Connectors that support security filtering previously used separate fetchers for content and access control. One fetcher type is now used for both content and security fetching. `AccessControlFetcher` has been deprecated. Fetcher implementations that use `AccessControlFetcher` should instead use `ContentFetcher`.

Bug fixes

Fusion

- Index Workbench search now shows Custom Stages uploaded through the blob store without requiring a refresh.
- New Query Workbench stages set to OFF will now stay off when applied.
- Fixed a bug where the collection picker in Query Workbench would retain a new collection even when canceling the operation.
- Fixed a bug where Query Workbench would sometimes require typing the full field name before selecting it.
- The column tick icon in the Rules UI no longer moves up when adding multiple synonyms.
- Quick start message window will now be shown after initial login even when licensing is missing.
- Fixed issue in the Helm QnA chart where having no defined secrets would break the Argo workflow.
- Added GCS and Azure Hadoop fs libraries to the Spark Kubernetes configuration to successfully run associated data loading scripts.
- Fixed a bug where aggregating signals from external catalog tables would ignore timestamps and cache all data.
- Added support for Query Pipeline operations on grouped responses with multiple groups. Old API methods that worked with single group fields, but would return unpredictable results if the response contained more than one field, are marked as deprecated.
- Updated Chrome driver dependencies versions to fix Web connector crashes.
- The job launcher now includes untracked Argo jobs to correctly report Argo status for situations where the job-launcher pod is killed and restarted.
- Reduced Spark pod to 30 characters to prevent running over the Kubernetes DNS pod name limits.
- Fixed a job service bug where multiple same-request interceptors executed and caused multiple internal redirects.
- Upgraded several services to the latest version of Jersey libraries.
- Error messages in the datasources panel will now show the error icon.
- Improved form autofill to display new values instead of default after saving.
- Collections for an app no longer show _recommender_models collections from other apps.
- Fixed console error when closing the Parsers pane in the UI.
- Improved Index Pipelines in the UI to not allow adding stages with duplicate IDs.
- The Messaging Service dropdown will now show an actual fallback value instead of 'undefined'.
- QnA pipeline dropdowns now include tooltips to account for long names.
- Cold Start Training job field is now marked as required.
- Pipelines dropdown is now sorted on datasource page.
- Removed false console warning when multiple IDs exist in the scheduler.
- Updated facet label management to allow creating and editing blobs from the Query Workbench.
- Updated links to documentation within the Admin UI.
- Query Workbench rows parameter override feature fixed so QWB results in Firefox reflect JavaScript stage changes.
- Metrics for individual stages are no longer lost when those stages are asynchronous.
- Added liveness probe for Pulsar broker deployments that attempts to detect brokers in the bad state and restart them automatically.

- Collections that are not associated with the current app are now searchable in Object Explorer.
- Fixed issue where system autocomplete collection was not created without manual intervention if an error occurred during the first attempt to create it.
- Fixed an issue with the fusion-notebooks service chart where specifying resource requests and limits would cause the install to fail due to invalid YAML syntax.
- Azure connector now restarts if initially missing Storage Container field.
- Restored 4.x ability to send Arrays to Solr as multivalued objects while also allowing translation of nested JSON structures.
- Access table now sorts usernames in descending alphabetical order.

Fusion

- Fixed a bug where the tooltip would cover the delete option for individual synonyms.
- Fixed a bug where the synonyms cell would not display newly created synonyms.
- Updated liveness/readiness probes in ml-model service to be configurable.

Predictive Merchandiser

- Fixed rule sorting to be consistent across rules with the same created date so that adding synonyms to a rule using search no longer resets the UI.
- Renamed facet labels in the Fusion Query Workbench stage now display the new name instead of the default field name.
- Fixed a bug where the 'created on' date was not fully visible.
- Improved speed when publishing a rule on a deployed cluster using the Publish button. A new parameter was added to specify the time interval for skipping Solr optimize for the `{{publish}}` end point. The makes the API call performance 6-7 times faster.
- Ability added to toggle which rules are fired via tags. Search rules are now visible in merchandiser views with associated tags.
- Fixed a bug where pagination controls were sometimes unavailable when total doc count was less than 20.
- Fixed a bug where the field typeahead value would sometimes show an empty suggestion list.
- Rules table now refreshes after creating a clone rule so cloning triggers a re-search.
- Item-to-item recommendations now display on the PDP in PM.
- Enabled entering field names and values which do not exist in the schema while editing a rule.
- Range facets will now display properly in PM
- Tag tooltip description now appropriately reflects the feature’s functionality.
- In previous releases, triggered BuryList rules would add the buried docs to the resultset even if they were not matched by the user’s original query. This change alters this behavior to be more intuitive - buried documents have their score reduced if they are in the result set but are not otherwise added.
- Delete button is hidden when a rule does not exist.
- Phrase Word Counter is now updated on the fly to account for edited phrases.

Connectors

- Fixed a problem with the V2 connector service downscaling during a running job.
- Fixed a bug which caused the V2 Connectors service to crash while crawling `FetchInputs` that produce many (that is, millions) of documents. Crawling large files now finishes indexing the documents as expected.
- Index profile panel can now open when connectors-classic service is not running.

- The UI no longer reports an error if connectors services are not deployed in the cluster.

Known Issues

Fusion

Imported and exported apps do not include subscriptions. This issue is fixed in Fusion 5.2.

Connectors

- Prometheus metrics do not work for the Rest or Classic connector services. This issue is fixed in Fusion 5.2.1 and 5.3.
- The *FTP connector* displays an error and does not run in 5.1.2. This issue is fixed in Fusion 5.1.3 and 5.2.

Predictive Merchandiser

- Rules with disallowed symbols do not appear when the **Show Applied Rules** button is clicked in the **Details** view. This issue is fixed in Fusion 5.2.1 and *5.3*.
- When a conflicting bury rule is added to a boosted product, it changes other boosted product positions. This issue is fixed in Fusion 5.2.

Fusion 5.1.1

Release date: April 8, 2020

Component versions:

Component	Version
Solr	8.4.1
ZooKeeper	3.5.6
Spark	2.4.3
Kubernetes	GKE, AKS, EKS 1.18 Rancher (RKE) and OpenShift 4 compatible with Kubernetes 1.18 OpenStack and customized Kubernetes installs not supported. See *Kubernetes support* for end of support dates.
Ingress Controllers	Nginx, Ambassador (Envoy), GKE Ingress Controller Istio not supported.

More information about support dates can be found at *Lucidworks Fusion Product Lifecycle*.

Docker image versions:

```
apachepulsar/pulsar-all:2.5.0
argoproj/argoui:v2.4.3
argoproj/workflow-controller:v2.4.3
bitnami/kubectl:1.15-debian-9
busybox:1.31.1
busybox:latest
curlimages/curl:7.68.0
docker.io/seldonio/seldon-core-operator:1.0.1
grafana/grafana:6.6.2
jimmidyson/configmap-reload:v0.3.0
lucidworks/admin-ui:5.1.1
lucidworks/api-gateway:5.1.1
lucidworks/auth-ui:5.1.0
lucidworks/classic-rest-service:5.1.1
lucidworks/devops-ui:5.1.0
lucidworks/fusion-api:5.1.1
lucidworks/fusion-indexing:5.1.1
lucidworks/fusion-logstash:5.1.0
lucidworks/insights:5.1.0
lucidworks/job-launcher:5.1.1
lucidworks/job-rest-server:5.1.1
lucidworks/ml-model-service:5.1.0
lucidworks/ml-python-image:5.1.0
lucidworks/pm-ui:5.1.0
lucidworks/query-pipeline:5.1.1
lucidworks/rest-service:5.1.1
lucidworks/rpc-service:5.1.1
lucidworks/rules-ui:5.1.0
lucidworks/webapps:5.1.0
prom/prometheus:v2.16.0
prom/pushgateway:v1.0.1
quay.io/coreos/kube-state-metrics:v1.9.5
quay.io/datawire/ambassador:0.86.1
seldonio/seldon-core-operator:1.0.1
solr:8.4.1
zookeeper:3.5.6
```

Looking to upgrade?
Check out the *Fusion 5 Upgrades topic* for details.

Improvements

Fusion

- Previously, subscriptions were deleted by referencing a unique, auto-generated ID. Now, subscriptions can be deleted by calling a new endpoint using this structure:

  ```
  DELETE /subscriptions/resource={resourceType:resourceName}&topics=
  {topicsArray}&context=app:{appName}
  ```

  For example, to delete subscriptions for topics `topic1` and `topic2` for an index pipeline named `myIndexPipeline` in `my_eComm_app`:

```
DELETE /subscriptions/resource={index-pipeline:myIndexPipeline}&topics=
{topic1,topic2}&context=app:{my_eComm_app}
```

- The Pulsar default namespace is now derived from create/update operations in an index pipeline or profile. Previously, this was done when the message producer stage was first enacted.

Bug fixes

Fusion

- When subscriptions are refreshed, all nodes now receive the action to either start or stop the subscription. Previously, only the original node refreshed.
- Updated Feign Clients for Spark pipeline indexing to reduce failures.
- Corrected the source of configuration values for deploy jobs so the values are taken from the Helm chart.

Known issues

Fusion

- Imported and exported apps do not include subscriptions. This issue is fixed in Fusion 5.2.
- The rules table does not refresh after creating a clone rule. *This issue* is fixed in Fusion 5.1.2 and 5.2.
- The *Document Clustering job* can fail if a `trainingDataFilterQuery` is used. *This issue* is fixed in Fusion 5.2.

Predictive Merchandiser

- Facet names do not appear in Predictive Merchandiser if the facets were created in a query pipeline using the *Field Facet stage*. *This issue* is fixed in Fusion 5.2.
- *Item-for-item recommendations* are not displayed in the **Product Details** panel. This issue is fixed in Fusion *5.1.2* and *5.2*.
- The **Field Values** drop-down in the **Edit Rule** menu is populated with sequential integers, preventing users from typing a value. This issue is fixed in Fusion 5.2.

Connectors

- The Web connector crashes and displays an "Invalid Session ID" error. *This issue* is fixed in Fusion 5.1.2 and 5.2.
- Jobs status icons intermittently fail to update. *This issue* is fixed in Fusion 5.2.

Fusion 5.1.0

Release date: March 12, 2020

Component versions:

Component	Version
Solr	8.4.1
ZooKeeper	3.5.6
Spark	2.4.3
Kubernetes	GKE, AKS, EKS 1.17 Rancher (RKE) and OpenShift 4 compatible with Kubernetes 1.17 OpenStack and customized Kubernetes installs not supported. See *Kubernetes support* for end of support dates.
Ingress Controllers	Nginx, Ambassador (Envoy), GKE Ingress Controller Istio not supported.

More information about support dates can be found at *Lucidworks Fusion Product Lifecycle*.

	Looking to upgrade? Check out the *Fusion 5 Upgrades topic* for details.

New features

Fusion

Index Stage SDK

Lucidworks now provides an *Index Stage SDK* in a public repository on GitHub *(https://github.com/lucidworks/index-stage-sdk)* with all the resources you need to develop custom index stages with Java.

Clone the repository to get started:

```
git clone https://github.com/lucidworks/index-stage-sdk
```

See Gradle quickstart *(https://docs.gradle.org/current/userguide/tutorial_java_projects.html)* documentation for more information on Java Projects.

New V2 connectors

Three new *V2 connectors* are released:

- ***Couchbase V2*** – The Couchbase V2 connector uses the Couchbase Java client to retrieve data stored in Couchbase *(https://docs.couchbase.com/java-sdk/current/hello-world/start-using-sdk.html)*.
- ***Sharepoint V2*** – The SharePoint connector retrieves content and metadata from an on-premises SharePoint repository. The new connector includes many *benefits* associated with V2 connectors.

- ***Active Directory Connector for ACLs V2*** – The Active Directory Connector for ACLs (V2) indexes Access Control List (ACL) information into a configured "sidecar" Solr collection, so that it can be used by other connectors.

Apache Pulsar

Fusion 5.1.0 adds Apache Pulsar, a message bus that combines queuing with publishing-subscribing features for server-to-server messaging. A producer publishes a message to a topic and a consumer subscribes to that *topic* using a *subscription type* to receive the topic messages.

Highlights include:

- Supports native streaming transformations through Pulsar Functions
- Uses Apache Zookeeper for configuration management
- Uses Apache BookKeeper for durable storage
- Each document processed has a result object persisted to another topic

Read more about *Apache Pulsar in Fusion*.

More new features

- A new metrics report Java library improves the functionality of the *DevOps center*.
- Fusion creates a log file for login attempts by producing an audit file with the API gateway.

Fusion

Sentiment analysis

One major part of understanding intent is understanding sentiment. When you assign sentiment values to incoming queries, or to documents during indexing, you can leverage that data to deliver more insight to your business applications.

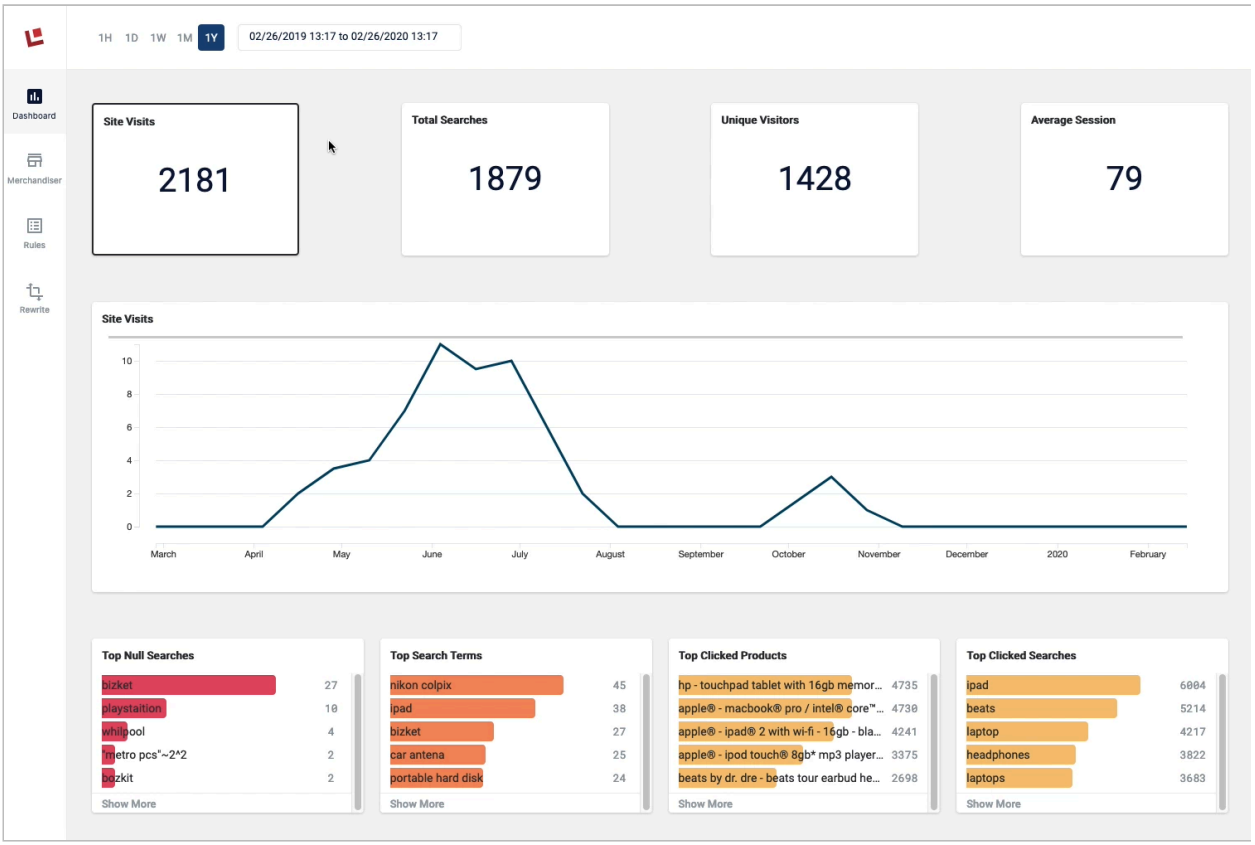
This release includes two state-of-the-art, pre-trained sentiment analysis models that you can use to embed sentiment scoring into your query and indexing pipelines:

sentiment-general	**sentiment-reviews**
A general-purpose model, trained on short sentences. Suitable for short texts and for intent prediction. See *Deploy The sentiment-general Model* for instructions.	A model trained on a variety of customer reviews. Optimized for longer texts. It also supports highlighting the tokens that provide stronger sentiment. See *Deploy The sentiment-reviews Model* for instructions.

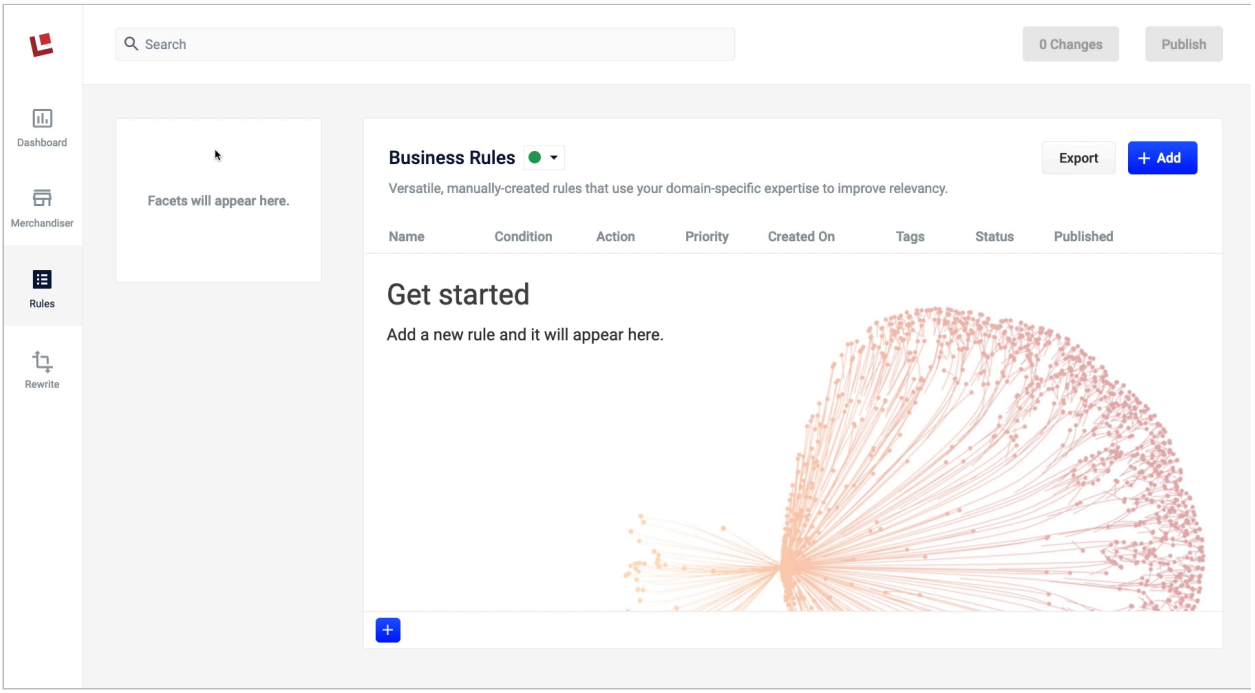
See *Sentiment Analysis and Prediction* for more details and instructions.

Predictive Merchandiser

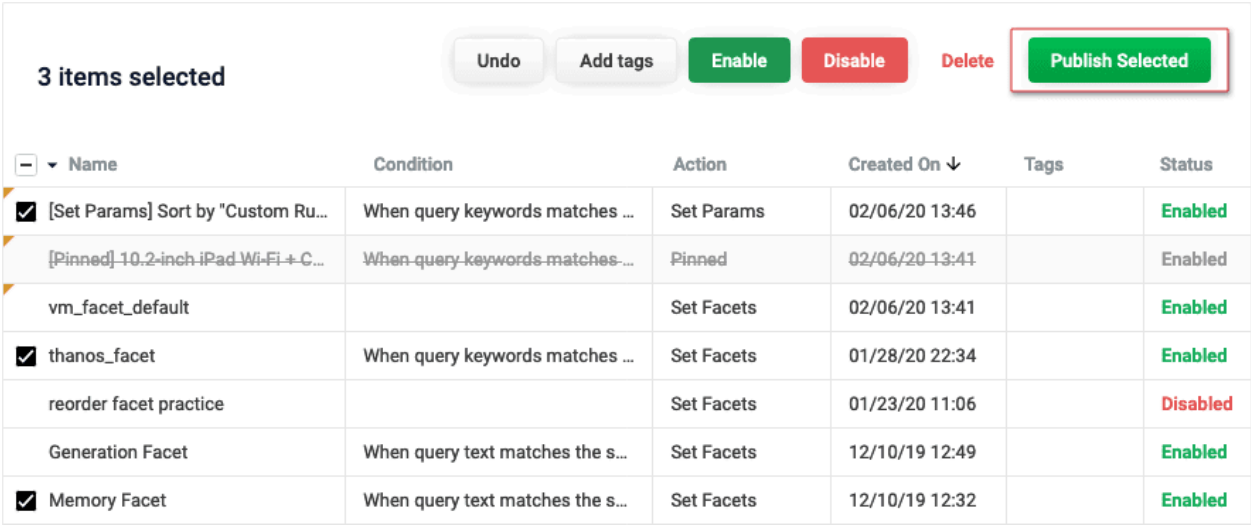
- Clicking the tiles in the Analytics screen now updates the graph below to display detailed metrics.



- The ability to switch apps from within the Rules Editor is added:



- Support is added for individually selecting rules to publish in the Fusion UI:



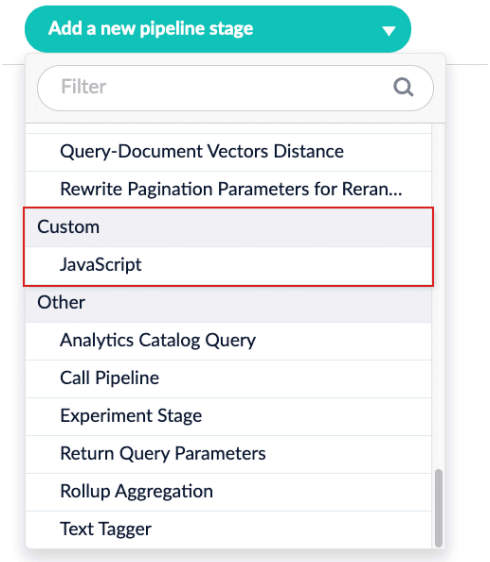
Improvements

Fusion

- Index stage plugins can now be uploaded directly to the *Blob manager*. To upload to the *API* with a PUT request, use the `api/index-stages/plugins` endpoint. To delete the plugin, use a DELETE request with the `api/index-stages/plugins/<pluginId>` endpoint.
- The default realms created when Fusion first starts can be overridden using the `security.initial-realm-configs` spring boot properties. For example:

```
security: initial-realm-configs: { "realmType": "jwt", "name": "jwt_okta",
"enabled": true, "roleNames": [ "admin" ], "config": { "autoCreateUsers": true,
"jwtIssuer": "https://dev-362383.okta.com/oauth2/default", "jwkSetUri":
"https://dev-362383.okta.com/oauth2/default/v1/keys", "groups": { "groupKey":
"fusion-groups", "roleMapping": [ [ "admin", "admin" ] ] } }}
```

- The Index Workbench now supports "save as" functionality, like the Query Workbench.
- Users can now import Solr synonym definitions into Fusion through the *Blob Manager*.
- A new category, **Custom**, is added to index and query pipelines. Custom index pipelines include External collection query, *JavaScript*, and Simple. Custom query pipelines include Javascript.



- The `externalTrafficPolicy` for the api-gateway service is now set to `Local` by default. Existing users can configure this property by adding it to their `custom_fusion_values.yml` file:

```
api-gateway: externalTrafficPolicy: "Local"
```

- Trusted realm supports IPv6 normalization and netmask in CIDR notation.
- With Solr 8.x and 7.7.y, the `SOLR-OPTS` environment variable, found in the Java properties, can modify the `max.booleanClauses` variable.

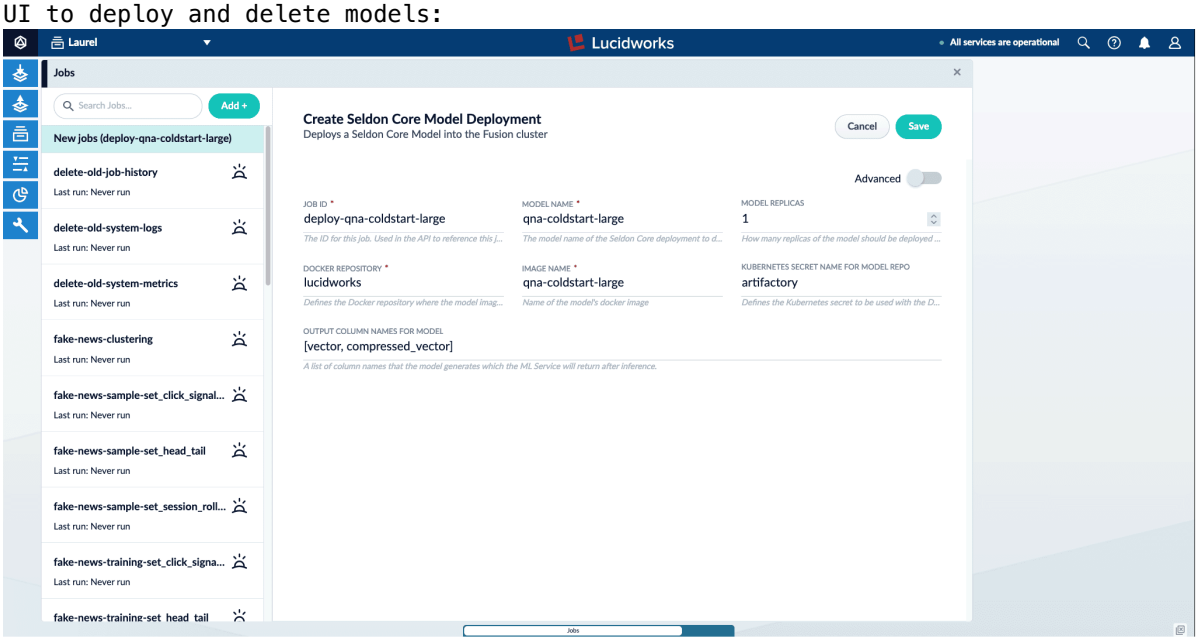
```
solr: image: solr:8.3 container_name: solr hostname: solr environment: ZK_HOST: zk:2181 SOLR_OPTS: -Dsolr.max.booleanClauses=5000
```

- The error message generated by sending click signals to a collection that has signals disabled is improved.
- Users with the role of "developer" now have expanded UI access.
- Various UI improvements are made.

Fusion

- Major updates to the Data Science Toolkit Integration. Model Devops made easy

- Integration with open Seldon Core *(https://docs.seldon.io/projects/seldon-core/en/latest/)* for custom machine learning model deployment in Fusion.
- Follows open source standards for model creation.
- Configure the Machine Learning *index stage* and *query stage* to interact with your deployed models.
- Each model is containerized individually as a Docker image, allowing independent deployment with no inter-model framework dependencies.
- Independent scaling of individual models.
- UI to deploy and delete models:



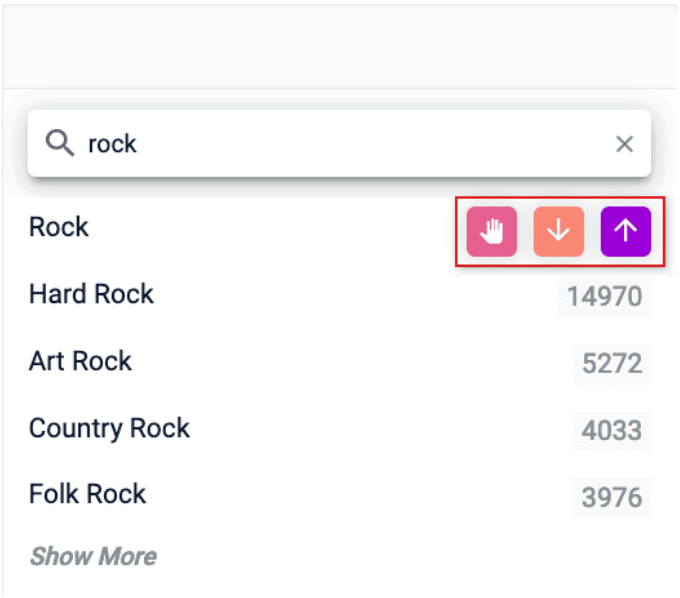
See *Develop and Deploy a Machine Learning Model* and the new *Create Seldon Core Model Deployment* and *Delete Seldon Core Model Deployment* jobs.

- A new parameter, `loadOnStartup`, is added for machine learning models. Models with `loadOnStartup=true` will be automatically loaded when the machine learning service is started or the model is loaded into the machine learning service. Default behavior requires the models to be loaded on demand.

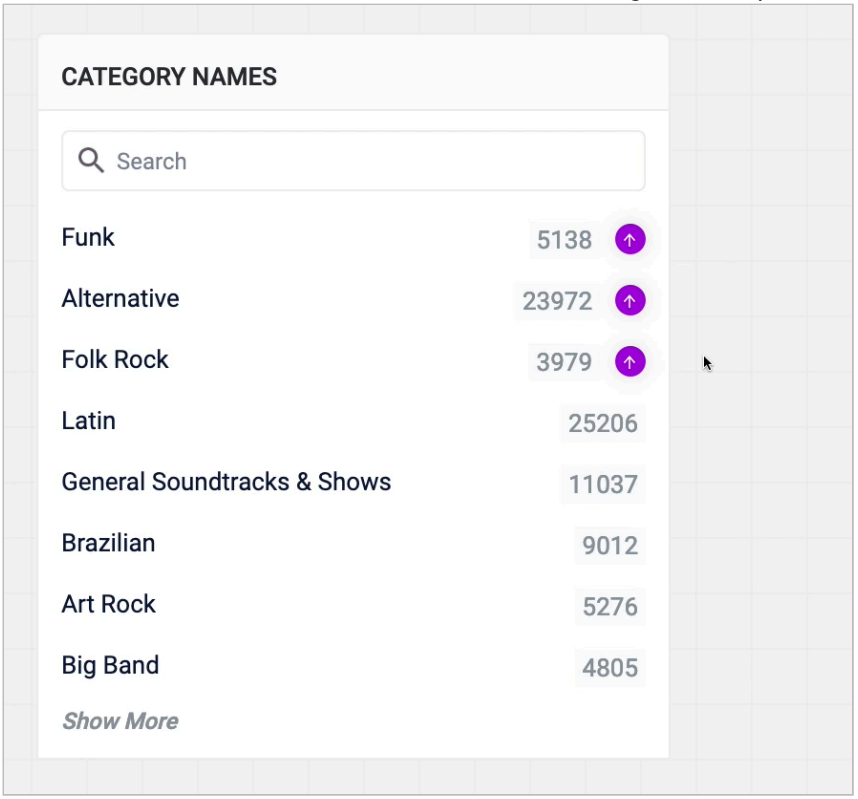
Predictive Merchandiser

Facets

- Remove, boost, bury, or block facets by hovering over individual facets. See *Use Predictive Merchandiser Facets*.
- Boost, bury, or block facets from your search results:



- Boosted facets can now be reordered with drag-and-drop:

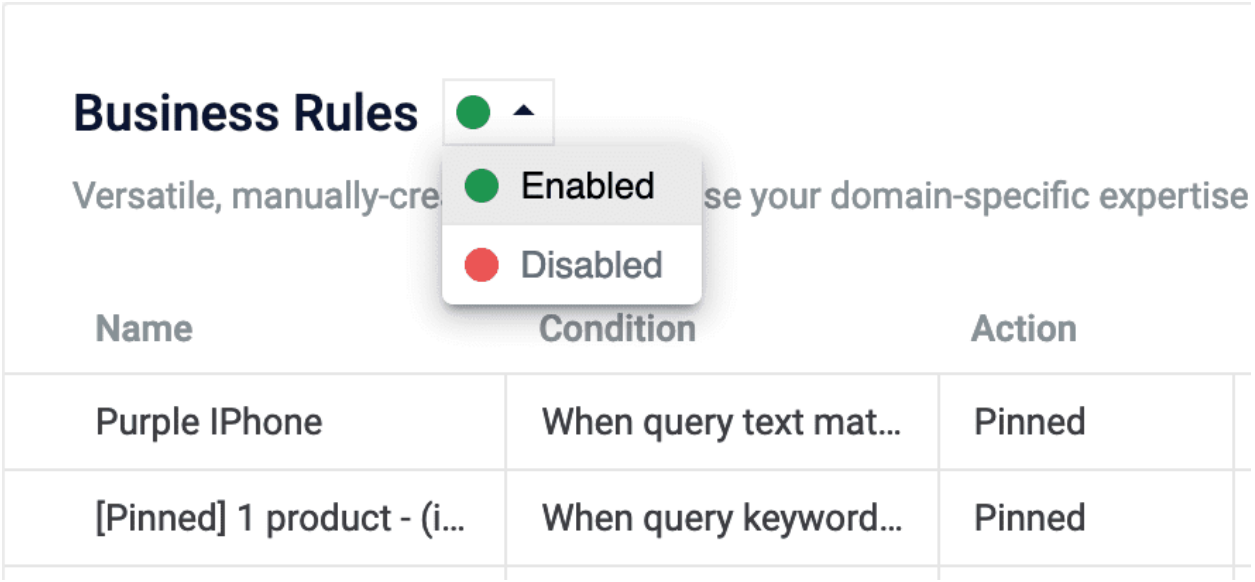


- Rule creation now includes selected facets as part of the rule.
- Users can now block facet filters for facets created in the Rules Editor.

Rules

- When creating a rule, active facets become part of the rule.
- When creating a new rule for a term that is associated with a query rewrite rule, the new rule will apply to the rewritten term.
  For example, if a query rewrite rule exists to rewrite the query `Mac` as `Macbook Pro`, creating a new rule for `Mac` will automatically apply the rule to `Macbook Pro` instead.
- You can now assign a new rule to each of your product pins. For example, you can pin products based on a sales promotion, holiday event, etc.
- Sort rules by priority with drag-and-drop, just as you can with facets.

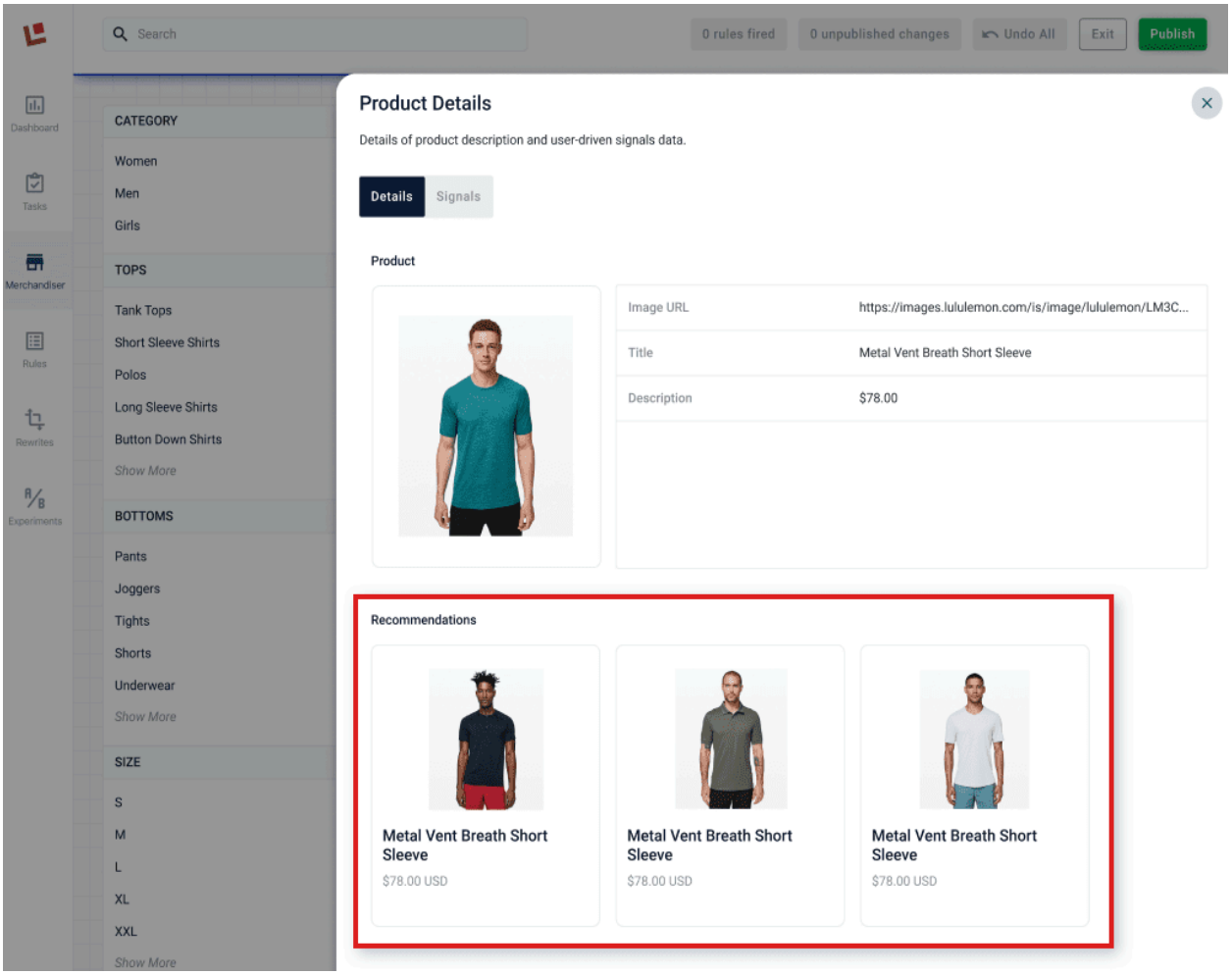
- You can now publish approved rules individually, instead of waiting for all pending approvals to be processed.
- Rules created for a query rewrite now apply to the rewrite instead of the original term.
- Added an optional **Frequency** column to the Query Rewrites dashboard for Phrases and Synonyms only, which you can use to sort and focus on high-value rewrites. You can add this column by customizing the columns for those rewrite types.
- Added `Reason` meta-data for head/tail rewrites, to explain the reason why the rewrite was suggested.
- Added `Type` meta-data for spell corrections rewrites, to explain whether a token-level correction or phrase-level correction is recommended.
- Added a new option for users to enable or disable query rewrite rule types.



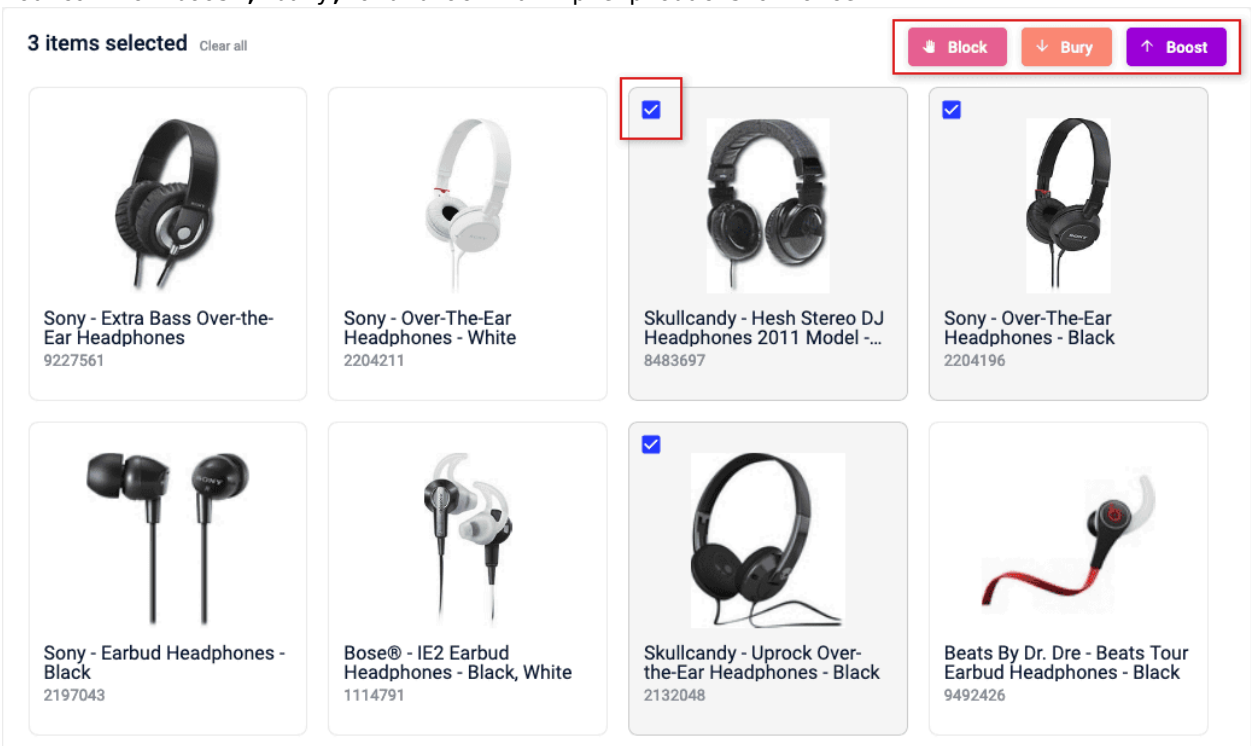
- Rules with high confidence ratings can be published automatically to production from the AI Jobs. Fusion 5.1 does not auto-publish from AI Jobs by default.

Other improvements

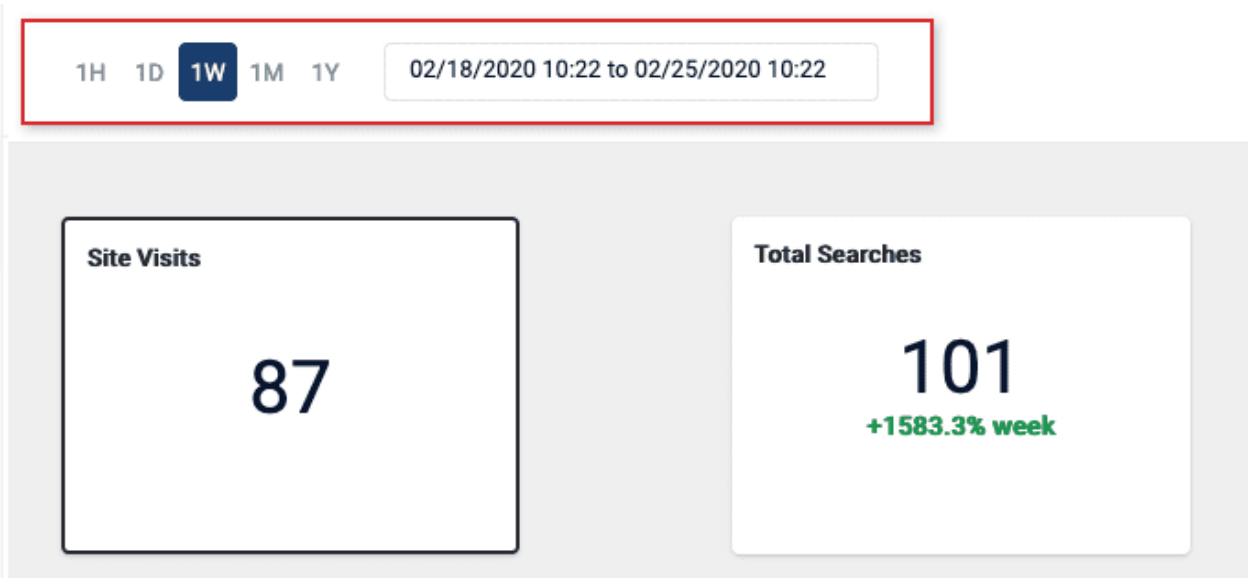
- Recommended products that have been generated by signals are included in the **Details** panel.



- You can now boost, bury, and block multiple products at once.



- Documents now feature an icon for drag-and-drop functionality.
- A date range filter component is added to the **Dashboard** for quickly sorting analytics data:



- Various UI improvements.

Addressed security vulnerabilities

Fusion

- Fixed an issue with a component in the `admin-ui` package that resulted in an security alert.

Bug fixes

Fusion

- The Parallel Bulk Loader (PBL) will no longer fail when a document fails to be uploaded to the index pipeline using `POST`. Enable this functionality in the PBL configuration.
- The Tika parser now honors MIME type parameters, for example: `{{application/vnd.wordperfect; version=5.0}}`. Previously, content MIME types parameters were not always correctly matched to types supported by Tika. This resulted in the Tika parser not recognizing content as parseable.
- Fixed a bug that prevented files with non-ascii filenames from being uploaded to the index workbench.
- Fixed a bug that sometimes prevented renamed files from showing up in query results after being reindexed.
- Fixed a bug that caused the query workbench to fail to load if there were over 10,000 Solr fields.
- Fusion no longer blocks access to the Blob Store and Collections Manager when the collections service is down.
- The Github connector no longer authenticates with a username and password, as this functionality is deprecated. Now, a personal access token is required.
- Connectors can now join a running job.
- Fixed a bug that sometimes resulted in an `ArrayIndexOutOfBoundsException` error when running a spell correction job.
- Fixed a bug that prevented apps created in Fusion 4.2.x from being imported to 5.y.
- All `zkConnectionStrings` values can be set with a single global variable, `zkConnectionString`. Previously, the values were set individually per service.
- Jupyter sessions will no longer expire while the user is still active.
- Fixed a bug with the de-duplicate feature that allowed spurious fields to be written to Solr.

- Fixed a bug that required a non-empty, non-root context path for deploying an application into the webapps service.

Fusion

- Switching between apps in the Fusion UI while viewing the query rewrite tab will no longer bring you to the rules tab.
- When multiple rules are selected in bulk, the enable and disable buttons are now inactive.
- Fixed a bug that prevented faceting query rewrites in Internet Explorer 11.
- Fixed a bug that disabled the option to delete query rewrite synonyms if a synonym was edited after being added.
- Fixed a bug that prevented the user from publishing rules by type.
- Fixed a bug that prevented query rewrite from creating multi-word synonyms.

Predictive Merchandiser

- Fixed a bug that prevented facet list items from reappearing when unselecting a facet category.
- Fixed a bug with the typeahead functionality of the search box.
- Fixed a bug that prevented items in the details table from appearing.

Known issues

Fusion

- When subscriptions are refreshed, only the original node receives the request to start or stop the subscription. This is fixed in *Fusion 5.1.1*.

Fusion 5.0.2

Release date: December 18, 2019

Component versions:

Component	Version
Solr	8.3.1
ZooKeeper	3.5.6
Spark	2.4.3
Kubernetes	GKE, AKS, EKS 1.18 Rancher (RKE) and OpenShift 4 compatible with Kubernetes 1.18 OpenStack and customized Kubernetes installs not supported. See *Kubernetes support* for end of support dates.
Ingress Controllers	Nginx, Ambassador (Envoy), GKE Ingress Controller Istio not supported.

More information about support dates can be found at *Lucidworks Fusion Product Lifecycle*.

	Looking to upgrade? Check out the *Fusion 5 Upgrades topic* for details.

New features

Predictive Merchandiser

Fusion 5.0.2 integrates Predictive Merchandiser as a component of *Fusion*.

If you have a *Fusion license*, you can access Predictive Merchandiser by navigating to **Relevance > Rules** and selecting **Merchandiser**.

To learn more, see *Predictive Merchandiser*.

Jupyter Integration

Starting with Fusion 5.0.2, we now provide a Jupyter service that can be run from the Fusion Helm chart. Jupyter *(https://jupyter.org/)* is an open-source web application that allows you to create and share documents that contain live code, equations, visualizations, and narrative text.

- Run/Debug Spark code in Scala/Python (replacement for spark-shell)
- Run SQL queries via Fusion SQL
- Debug Scala and SQL transforms in PBL jobs
- Everything else for which Jupyter is designed

To learn more, see *Jupyter Support in Fusion*.

Improvements

- The HTML parser now parses the `application/xhtml+xml` media type by default.
- The HTML parser can filter HTML content using jsoup *(https://jsoup.org)*. New HTML parser configuration options include:

Configuration option	Default value	Description
`excludeFilters`	N/A	Jsoup-formatted selectors for elements to exclude from the HTML
`filterBeforeMapping`	False	Apply exclude filters before performing HTML field mapping
`filterBeforeExtractingLinks`	False	Apply exclude filters before performing link extraction

- Index pipelines can now receive nested JSON documents through the Parallel Bulk Loader.
- The Fusion UI now supports **undo** for deleted rules.
- A new boolean property, `shareState`, is introduced for Javascript Query Stages. When set to `true`, a single JavaScript engine will be used for all queries on a Javascript query stage. Previously, a unique engine was used for each thread.
  Under certain conditions, this can improve performance significantly.
- If a rule creation API is incorrectly formatted in JSON, Fusion now produces a detailed error message.
- Improved the output from phrase extraction jobs. It is now more useful for non-signals datasets.
- Improvement made in error propagation between services.
- The Windows Share SMB 2/3 Connector now parses data into individual Solr fields instead of a large JSON document.
- The Custom Spark job has been replaced by a *Custom Python job*.
- Various UI improvements and fixes.

Bug Fixes

- Fixed a bug that sometimes resulted in the `APP_NAME_recommender_models` collection to delete the `APP_NAME_recommender_models` collection.
- The `collection_alias` field is now required for time-based partitioning in the Fusion SQL service.
- The roles selector in the **New User** panel now correctly displays the available role types.
- Fixed a bug that prevented the Box Connector from correctly indexing PowerPoint files.
- Fixed a bug that prevented the Box Connector from indexing Japanese text with UTF-8 character encoding.
- Fixed a bug that prevented V2 connector jobs from succeeding if a fetcher was defined without a phase name.
- Fixed a bug that produced an Out of Memory (OOM) error when indexing large JSON files with a V2 connector.
- V1 connector stateful jobs are no longer dependent on server IPs. recrawling after restarting V1 connectors now works as expected.
- Fixed a bug that removed CJK characters from file names when uploading a file.
- Fixed a bug the prevented the Jobs API from scheduling a recently created job.
- Fixed a bug that sometimes caused `DefaultHostInfoMetricReporter` to fail on Java 11.
- Fixed a bug that caused *head/tail analysis jobs* to overwrite published rules.
- Boost lists will now utilize the correct term, `orig_score`, when rewriting a query.
- Corrections made to the default synonym detection job settings.
- Fixed a bug that sometimes caused pytests to fail when using the `fusion` user.
- Logstash is now enabled by default.

Known issues

Fusion

- The Query Workbench rejects multiple parameters with the same name. This issue is fixed in Fusion 5.3.
- If *High performance mode* is selected on the Web connector configuration, the crawl does not start. This issue is fixed in Fusion 5.1.0.
- Deploying an application into the Webapps service requires a non-empty, non-root context path. This issue is fixed in Fusion 5.1.0.
- Exported apps are created without `solrParams`, causing an error when importing. This issue is fixed in Fusion 5.2.
- Failed requests to the Azure Kubernetes Service fall into an endless loop, causing rate limit violations in Azure. This issue is fixed in Fusion 5.3.
- App Studio apps fail to run when deployed on a Fusion 5 cluster. This issue is fixed in Fusion 5.2.
- The Solr exporter uses the same affinity rules as the Solr server. This issue is fixed in Fusion 5.1.
- Publishing a rule on a deployed cluster is unusually slow. *This issue* is fixed in Fusion 5.1.2 and 5.2.
- The default `rows` parameter only allows 10 rules to be triggered, even if a greater number of rules meet the triggering condition. This issue is fixed in Fusion 5.1.
- No error is displayed if a sub-action fails while creating an app. This issue is fixed in Fusion 5.2.
- The API gateway displays a "Some Services Unavailable" or "Force-closing" error. This issue is fixed in Fusion 5.0.3 and 5.1.
- Importing an app displays a 502 error, intermittently. This issue is fixed in Fusion 5.0.3.
- The Webapps service does not work with Twigkit apps. This issue is fixed in Fusion 5.0.3.
- The Index Workbench shows a blank screen or "No Results" message. This issue is fixed in Fusion 5.1.
- JavaScript evaluation does not start when initiated. This issue is fixed in Fusion 5.1.

Predictive Merchandiser

- Items in the global bury list still show in search results. This issue is fixed in Fusion 5.1.2 and 5.2.
- The **Rules** menu in Predictive Merchandiser redirects to the Fusion App Launcher. This issue is fixed in Fusion 5.1.
- Publishing rules does not produce a request payload or publish the correct number of items. This issue is fixed in Fusion 5.1.

Fusion 5.0.1

Release date: October 10, 2019

Component versions:

Component	Version
Solr	8.2
ZooKeeper	3.4.13
Spark	2.4.3
Kubernetes	GKE, AKS, EKS 1.15 Rancher (RKE) and OpenShift 4 compatible with Kubernetes 1.15 OpenStack and customized Kubernetes installs not supported. See *Kubernetes support* for end of support dates.
Ingress Controllers	Nginx, Ambassador (Envoy), GKE Ingress Controller Istio not supported.

More information about support dates can be found at *Lucidworks Fusion Product Lifecycle*.

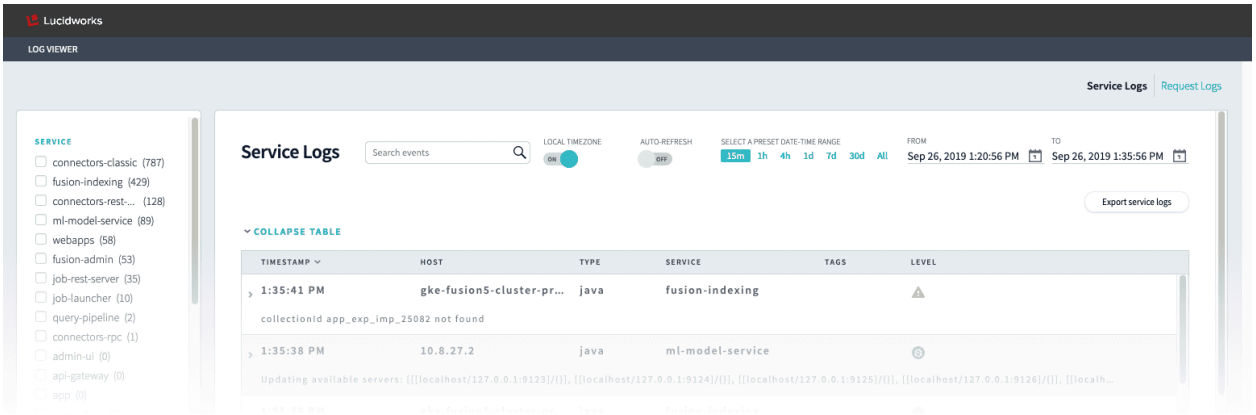
	Looking to upgrade? Check out the *Fusion 5 Upgrades topic* for details.

Improvements

- Performance is improved for the logging of the *SQL service*.
- Synonym detection is improved to better handle multi-word synonyms.

Other changes

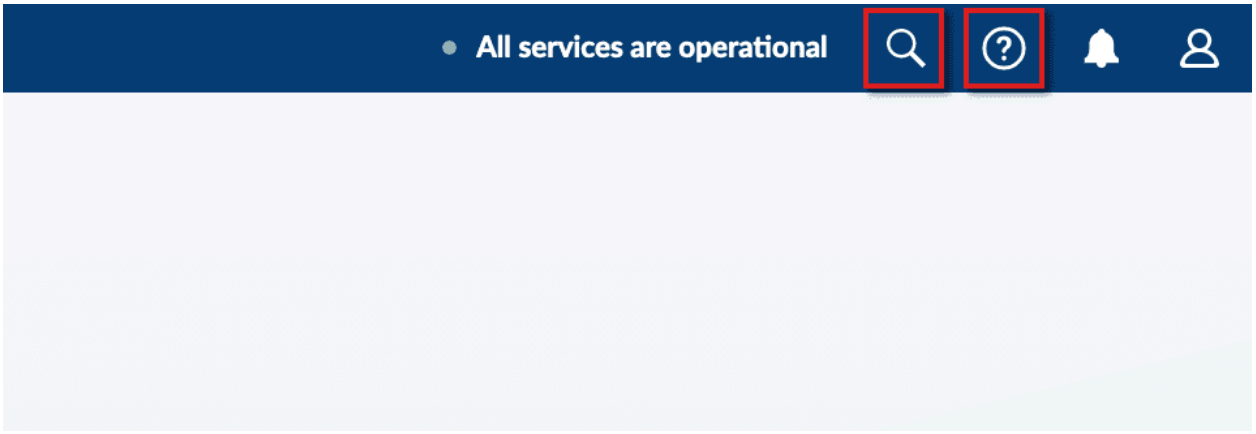
- The Cluster, Hosts, Services, and Datasources dashboards are removed from the DevOps center. The Log Viewer dashboard remains.



- Minor changes to the default Spark settings in Kubernetes.

Bug Fixes

- Fixed a bug that disabled parts of the **Query Pipelines > Add +** menu.
- Fixed a bug with the *Head/Tail Analysis Jobs* so that the `outputCollection` field is respected by the job.
- Fixed the default location of the Spark executor configurations `application-kubernetes.yml`. Configurations are now applied as expected.
- Fixed a bug with *SQL Aggregations* that caused the catch-up range from the second run to be used in subsequent runs.
- Fixed a bug that sometimes caused the *Token and Phrase Spell Correction Job* to fail with an `ArrayIndexOutOfBoundsException` error.
- Fixed a bug that caused WAR files to become disassociated from a webapp when the name of the webapp was changed.
- The docs search and link to the docs in the top nav now point to the correct version of the docs.



Known issues

Fusion

- The default field names in the Synonym Detection Job are listed incorrectly. This issue is fixed in Fusion 5.2.
- Setting the ML Model Service to use a Google Cloud Storage bucket for storage fails. This issue is fixed in Fusion 5.0.2.
- An error prevents the `set-params` rule from saving successfully when multiple parameters are defined. *This issue* is fixed in Fusion 5.2.
- Users must refresh the browser to view a new model after it is created in the Model Service. This issue is fixed in Fusion 5.0.2.
- The **Upload Datasource** function does not encode Chinese, Japanese, or Korean characters correctly. This issue is fixed in Fusion 5.0.2.
- Fusion displays an error when attempting to call a model in the ML classification job. This issue is fixed in Fusion 5.0.2.
- Running a Head/Tail Analysis job with an empty `tailRewriteCollection` displays an error. This issue is fixed in Fusion 5.0.
- **View Logs Dashboard** links in Fusion do not redirect to the error log or show the error in question. This issue is fixed in Fusion 5.1.
- Running any job type results in a retrieval error. *This issue* is fixed in Fusion 5.0.2.

Predictive Merchandiser

- `Created On` converts from datepicker to string and prevents changing the creation date when editing a rule. This issue is fixed in Fusion 5.1.2. and 5.2.

Connectors

- Requests to store HashMap metadata in Solr (crawlDB) fail. This issue is fixed in Fusion 5.1.
- Connector jobs no longer run if a pod IP that has been assigned to the job is removed and the job is not reset to a new pod. *This issue* is fixed in Fusion 5.0.2.

Fusion 5.0.0

Release date: September 11, 2019

Component versions:

Component	Version
Solr	8.2
ZooKeeper	3.4.13
Spark	2.4.3
Kubernetes	GKE, AKS, EKS 1.15 Rancher (RKE) and OpenShift 4 compatible with Kubernetes 1.15 OpenStack and customized Kubernetes installs not supported. See *Kubernetes support* for end of support dates.
Ingress Controllers	Nginx, Ambassador (Envoy), GKE Ingress Controller Istio not supported.

More information about support dates can be found at *Lucidworks Fusion Product Lifecycle*.

Lucidworks Fusion 5.0 lets customers easily deploy AI-powered data discovery and search applications in a modern, containerized architecture with built-in support for rapidly creating interactive dashboards and modern enterprise applications. Customers can leverage existing models and workflows, or create and deploy new ones quickly using popular tools like Python ML, TensorFlow, scikit-learn, and spaCy.

Kubernetes

Fusion 5.0 introduces Kubernetes (k8s) *(https://kubernetes.io)*, an open-source system for automating deployment, scaling, and management of containerized applications, as the recommended Fusion deployment model.

Enforcing best practices
With Kubernetes, Fusion users have a common language to declare how Fusion should be installed, configured, and maintained in production. Instead of focusing on internal implementation details, operations engineers will manage Fusion by monitoring how it uses native Kubernetes resources.

Reducing cost of ownership
Fusion with Kubernetes allows operations engineers to run Fusion, reducing the need for specialized training and internal experts.

Simplifying maintenance and upgrades
Kubernetes allows Fusion users to make zero downtime upgrades simple with rolling updates, minimizing the risks associated with changing a live cluster.

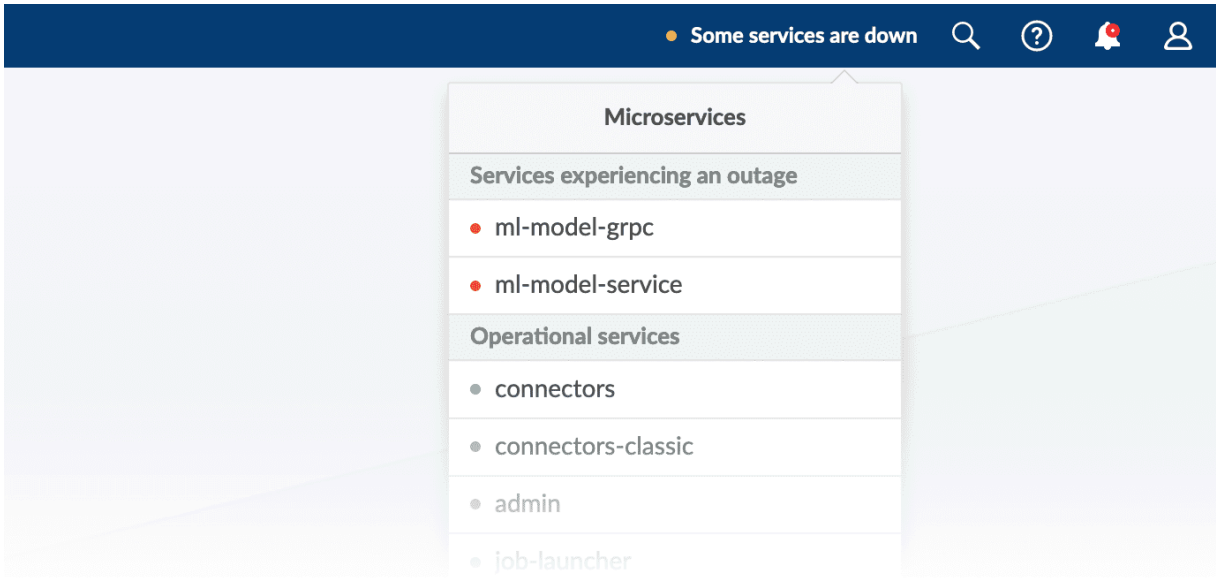
Spring Boot

Traditionally, the cross-cutting concerns of Fusion's various *microservices* have been handled with a chassis built and maintained by Lucidworks. Now, Fusion 5.0 employs Spring Boot *(https://spring.io/projects/spring-boot)* as its microservice chassis.

Whether Fusion 5.0 is being deployed in a Kubernetes, non-Kubernetes, or on-premises environment, Spring Boot allows Fusion to be deployed using the same underlying framework.

New features

- Fusion 5.0 expands the capabilities of the *Fusion SQL Service* query syntax significantly. Fusion SQL now offers:
  - Phrase queries: `x = 'Lucidworks Fusion'`
  - Sloppy phrase queries: `x = '("Fusion"~1)'`
  - Non-phrase queries: `x = '(Lucidworks Fusion)'`
  - LIKE operator queries: `x LIKE 'Luci%'`
- Fusion 5.0 introduces multiple usability improvements in the Fusion UI. Notable improvements include:
  - Long lists now load incrementally with a "Load More" button.
  - App microservices now have service indicators.



  - Stacktrace logs can now be downloaded directly from Fusion UI.
- Updates to the *Fusion SQL service* query operations change the default behavior of sorting query results by the documents' Solr score.
  - By default, query operations now return random samples of the matching results. Random sampling can also be utilized with `rand()`, for example: `select * from tableA order by rand() desc limit 5000`.
    This change makes it easier to use the `jdbc` Solr streaming expression *(https://lucene.apache.org/solr/guide/8_2/streaming-expressions.html)* to query Fusion SQL. Results from these queries can then be analyzed with Solr math expressions *(https://lucene.apache.org/solr/guide/8_2/math-expressions.html)*.
  - A User Defined Function (UDF), `score()`, has been introduced to allow users to sort by a document's Solr score. This was the previous default behavior. The UDF must be aliased to work correctly, for example: `SELECT id, score() as doc_score`.
- The SharePoint Online connector now supports app-only authentication, allowing you to associate searches with an app client id and reduce rate limiting.

Improvements

- In Fusion 5.0, Spark operates in native Kubernetes mode instead of standalone mode. See *Spark Administration in Kubernetes* to learn more about Spark operations in Fusion 5.0.
- Fusion 5.0 supports multiple instances of the *Fusion SQL service*.
- API services can be discovered and connected with external services. For example: `curl "http://localhost:8765/api/v1/objects/proxyApps"`
- In previous versions of Fusion, image metadata could not be parsed unless the images were parsed by enabling the **Include Images** option. Now, image metadata can be parsed and indexed separately from the images.
- A new API gateway, `/webapps`, is introduced for the purpose of deploying WAR files. For example, if a webapp `myapp.war` is deployed, it is accessible at `/webapps/myapp.war`.

- The *SharePoint Online connector* now allows Azure Active Directory bearer token authentication.
- The *Confluence connector* now supports Windows NTLM authentication.
- In the Google Drive connector, the `SERVICE ACCOUNT P12 PRIVATE KEY PASSWORD` field has been changed from a text field to a password field to improve security.
- The *HTML parser stage* now supports jsoup *(https://jsoup.org)* selector methods and can be configured to parse HTML or other data.
- Ignite-based security trimming can now be enabled or disabled on the *Box connector*.
- Fusion 5.0 adds more options for setting the status code from the query pipeline:
  - The JS response object supports a new property, `statusCode`, which can be set to the desired HTML status code (default `200`):

    ```
    response.statusCode = 429;
    ```

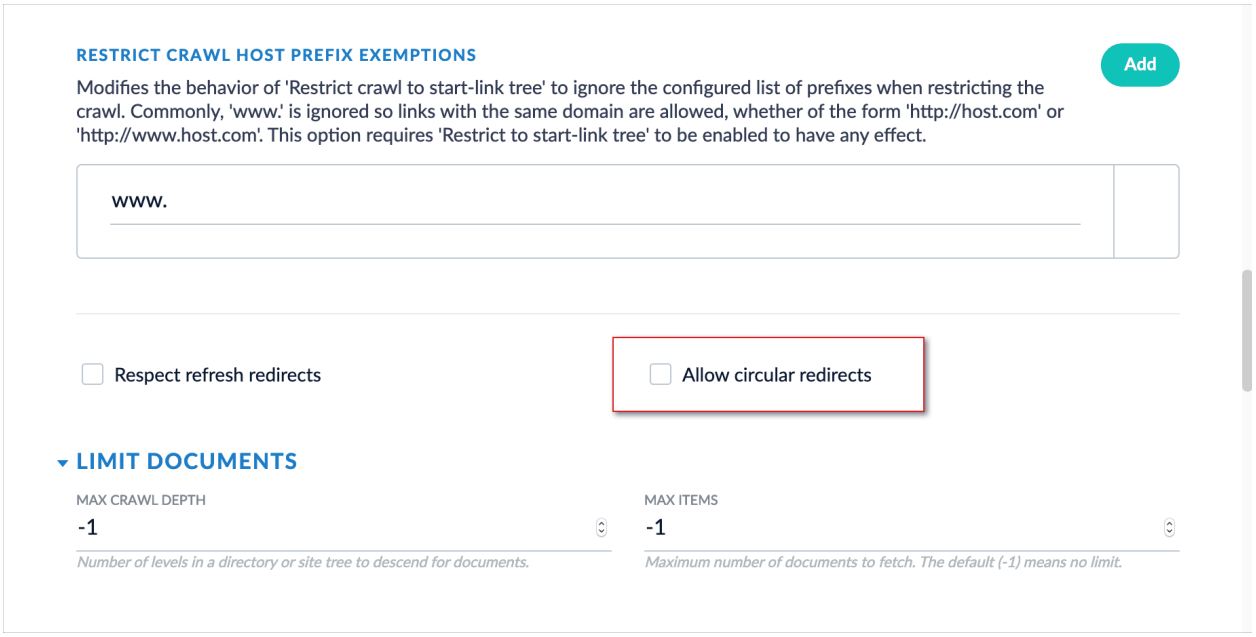
  - Use Jax RS exceptions to set the corresponding HTML status code from *custom JavaScript stages* for query pipelines:

    ```
    var WebApplicationException =
    Java.type("javax.ws.rs.WebApplicationException"); throw new
    WebApplicationException("Back off!", 429);
    ```

  - The *REST Query* index stage can now handle error responses and propagate status codes to a client:

    ```
    curl -u USERNAME:PASSWORD -X POST -H 'Content-type: application/json'
    'http://FUSION_HOST:FUSION_PORT/api/query-pipelines' -d '{ "id": "rpc",
    "stages": [ { "type": "query-rpc", "resultsLocation": "As Response",
    "errorHandling": "map", "params": { "uri": "http://localhost:8765/404",
    "method": "get" } }, { "type": "solr-query" } ] }' curl -v -u
    USERNAME:PASSWORD 'http://FUSION_HOST:FUSION_PORT/api/query-
    pipelines/rpc/collections/system_logs/select?q=*:*&rows=0' < HTTP/1.1 404 Not
    Found ... <title>Error 404 Not Found</title> ...
    ```

- Logging of killed nodes is improved by making the log message node-specific.
- The Web Connector now offers a new property, `circularRedirectsAllowed`, that toggles whether to block or allow circular redirects. This can be enabled in the UI:



- The way stopping connector jobs is handled is changed to reduce the amount of memory used.
- Updated `AbstractCrudResource.updateEntity` so that a PUT request creates an object if it does not already exist.
- *SMB V1 connector* query operations now allow for owner data to be retrieved as a metadata field.
- Waiting times at the end of connector jobs are reduced.
- Waiting times for creating new apps are reduced with changes to default jobs and schedules.
- Forms now support advanced fields and property groups, enabled by the toggle element **Advanced**.
- Fusion now isolates CookieStore values for each ChromeWorker to reduce the chance of overwriting cookies.
- Fetch performance when running Fusion on multiple nodes is improved.
- The *Sitecore connector* now supports token-based authorization.

Other changes

- The Heartbeat Data service is removed.
- The System Dashboard is removed, and its various components are moved.
- Admin webapps are removed.

Bug Fixes

- Query stage rule conditions of field value type now work regardless of format, instead of requiring a perfect match.
- The JSON parser now emits all documents when the `id` is substituted with another field.
- The SharePoint Online connector properly deletes all documents from deleted sites when the data is recrawled.
- Fixed an issue that caused queries in the *SQL Service* which had `0` results to produce an error.
- CrawlDB now stores data on-disk by default, as storing data in-memory was prone to crashes.
- Fixed an issue that caused an error to occur when executing a spell correction job when signals were missing or incompletely digested.

- The `collectMetricsIntervalSecs` parameter found in `/conf/fusion.properties` now works as expected.
- Predicate expressions on multi-valued fields in the *SQL Service* now work as expected.
- Fixed an issue with the SQL Service that caused an error to be produced when using `SELECT` with an unspecified limit for results. For example: `select id from myapp`.
- Changed the way the SDK plugin installation is handled in multi-node environments to reduce the chance of errors.
- Changed the way installing or updating plugins in a multi-node environment is handled to reduce the chance of errors.
- Fixed an issue that allowed users without the necessary access writes to upload blobs using `importPolicy=merge` and `importPolicy=abort`.
- Fixed an issue that caused the HTML parser to fail to parse an HTML document unless its `contentType` value was strictly `text/html`. For example, `contentType = "text/html; charset=utf-8"` would fail to parse.
- Fixed an incorrectly listed metadata field in the SharePoint connector, `Description_s`, so it used the correct field, `Description`.
- Fixed an issue in Windows 10 that caused the logs filter in the DevOps Center to be missing some fields and options.
- Fixed an issue that resulted in the SDK connectors service crawling all documents in a collection despite the maximum output count being set to `5`.
- Fixed an issue that caused the `connectors-rpc` service to fail if a plugin was deleted when the service was stopped.
- Fixed the Job History tab so all previous jobs are displayed.
- Fixed an issue that caused duplicate records to be produced by the *Google Drive connector*.
- Fixed an issue with Google Drive connector that prevented authorized users from overriding security trimming settings as desired.
- Fixed an issue with base64 encoding in the *Apache Tika Parser Stage*.
- The *JSON Parser Stage* now parses all values from a multivalued field as expected.
- Fixed an issue that caused an error to occur when a password-secured datasource was saved.
- Fixed an issue that occasionally caused some documents to be skipped due to an immense term error caused by a lengthy value in the `signature_s` field while indexing a SharePoint load test environment.
- Fixed an issue that caused a `NullPointerException` error to result when conducting SharePoint security trimming on a user with no assigned groups.
- Fixed an issue that caused case sensitivity for file extensions in connectors.
- Improved scaling for Phrase Detection jobs.
- Fixed an issue that caused an NPE error to be given when attempting to use the `/api/connectors/security_filter` endpoint without specifying a username.
- Fixed a compatibility issue between the Confluence connector and certain versions of Confluence.
- Fixed an issue with the SharePoint connector that sometimes caused URLs to become invalid when data was recrawled.
- Fixed an issue with the Confluence connector which resulted in anonymous access permissions not being updated as expected.
- Fixed an issue with the SharePoint connector that resulted in documents remaining in Fusion despite being deleted from SharePoint.
- Fixed an issue with the SharePoint connector that caused crawls to fail with a `NullPointerException` error if the crawl was attempted with no parser set.
- Fixed a bug that prevented regex searches from functioning unless *`root`* was specified as a value.

- Fixed a bug with the Apply Rules stage which forced Lucene to serve as the parser if boost, bury, block, or filter rules were used.
- Fixed a bug that prevented minimum match list rules from working as expected.
- Fixed an issue with the WebFetcher service that sometimes caused it to fail to emit multiple documents from the parser.

Known issues

- The *time-decay* function may not work as expected, depending on *rollup* settings.
- Fusion may not isolate collections between applications correctly within the UI.
- The Windows Share SMB 2/3 Connector's *Access Control Lists* may not be indexed due to a `size limit exceeded` error.

Deprecations and Removals

Active deprecations

The expected removal versions listed in the **Active Deprecations** table below may not correspond to a planned release version. Removals take place *on or after* the expected removal version. If the expected removal is listed as 5.2.2, but Fusion 5.2.2 is not a planned release, the removal may take place in Fusion 5.3.0.

> Fusion Connectors deprecations and removals
> Deprecation and removal details for Fusion Connectors are found at *Fusion Connectors Deprecations and Removals*.

Feature	Deprecated	Expected Removal	Notes
Fusion 5.12			
App Insights	5.12.0	5.13.0	–
Banana Dashboards	5.12.0	5.13.0	Users who require continued use can leverage a separate Docker deployment option. This alternative deployment method will be available in Fusion 5.13.0.
Parsers Indexing CRUD API	5.12.0	TBD	Use `/async-parsing/` instead.
Train a Smart Answers Cold Model Job	5.12.0	TBD	Use one of the pre-trained models or the supervised training job instead.
Data Augmentation Job	5.12.0	TBD	–
Fusion 5.11			
Webapps Service	5.11.0	5.13.0	Use *Deploy App Studio Enterprise to a Fusion 5 Cluster (GKE)* instead.
Fusion 5.9			
Security Trimming Query Stage	5.9.0	TBD	Use the *Graph Security Trimming Query Stage* instead, which uses a single filter query for all data sources.
Fusion 5.8			
Field Parser Index Stage	5.8.0	TBD	–
Tika Server Parser	5.8.0	TBD	–
Apache Tika Parser	5.8.0	TBD	–
Forked Apache Tika Parser	5.8.0	5.10.0	–
Fusion 5.6			
Hierarchical Filter in Apply Rules Query Stage	5.6.0	TBD	Use *Hierarchical field settings* in Predictive Merchandiser Templates.

Feature	Deprecated	Expected Removal	Notes
Fusion 5.2			
ALS Recommender Job	5.2.0	5.3.0	Use the experimental BPR Recommender job instead, which provides better results with a shorter running time.
Logistic Regression Classifier Training Jobs	5.2.0	5.3.0	Use the new Classification job instead, which provides more options and better logging.
MLeap deployments of SpaCy and SparkNLP	5.2.0	5.3.0	Use *Develop and Deploy a Machine Learning Model* instead.
MLeap in Machine Learning models	5.2.0	5.3.0	–
Query-to-Query Collaborative Similarity Job	5.2.0	5.3.0	Use the Query-to-Query Session-Based Similarity jobs instead, which provides better performance and query coverage.
Random Forest Classifier Training Jobs	5.2.0	5.3.0	Use the new Classification job instead, which provides more options and better logging.
Searchable aspects of the Blob Store	5.2.0	5.3.0	–
Time-based partitioning	5.2.0	5.3.0	–
Word2Vec Model Training Jobs	5.2.0	5.3.0	–
Fusion 5.1			
Connector fetcher `AccessControlFetcher`.	5.1.2	5.2.2	Use `ContentFetcher` instead.
Helm chart variable `solrAutocalingPolicyJson` for `fusion-admin`	5.1.2	5.2.2	Use `solrAutoscalingPolicyJson` instead. Update any custom values files that reference this variable.
Messaging Stage Configs	5.1.2	5.2.2	–

Removals

Feature	Removed	Notes
Fusion 5.11		
MLeap	5.11.0	Use *Develop and Deploy a Machine Learning Model* instead.
Subscriptions UI	5.11.0	–
Fusion 5.10		
Forked Apache Tika Parser	5.10.0	This parser was replaced by the *Tika Asynchronous Parser*.
Analytics Catalog Query Stage	5.10.0	–
Fusion 5.7		
NLP Annotator Index Stage	5.7.0	For an example you can modify for your needs, see *Develop and Deploy a Machine Learning Model*.
NLP Annotator Query Stage	5.7.0	For an example you can modify for your needs, see *Develop and Deploy a Machine Learning Model*.
OpenNLP NER Extraction Index Stage	5.7.0	For an example you can modify for your needs, see *Develop and Deploy a Machine Learning Model*.
Fusion 5.6		
Fusion SQL	5.6.1	This service was removed in 5.6.1. It is available for Fusion releases 5.5 and earlier.
Apache Pulsar	5.6.0	Kafka replaces Apache Pulsar.
Log Viewer & DevOps Center UI panel	5.6.0	These features used Apache Pulsar, which was replaced by Kafka.
Subscriptions API	5.6.0	–
Send to Message Bus Index Stage	5.6.0	–
Fusion 5.5		

Feature	Removed	Notes
Jupyter	5.5.2	–
Superset	5.5.2	–
Fusion 5.3		
Solr Clusters UI panel	5.3.0	–
Fusion 5.2		
Co-occurrence Similarity Jobs	5.2.0	–
Collection Analysis Jobs	5.2.0	–
Item Similarity Recommender Jobs	5.2.0	–
Legacy Item Recommender Jobs	5.2.0	–
Legacy Item Similarity Jobs	5.2.0	–
Levenshtein Spell Checking Jobs	5.2.0	–
Matrix Decomposition-Based Query-Query Similarity Jobs	5.2.0	–
REST API endpoint `/api/apollo/`.	5.2.0	Use endpoint `/api/` instead.
SQL-Based Experiment Metric Jobs	5.2.0	–
Statistically Interesting Phrases Jobs	5.2.0	–
Fusion 5.0		
The Cluster, Hosts, Services, and Datasources dashboards (DevOps center)	5.0.1	–

Feature	Removed	Notes
Admin webapps	5.0.0	–
Heartbeat Data Service	5.0.0	–
System Dashboard	5.0.0	–