



SPT Relational Database Web Tool
Application Manual
Version 1.1.2

Prepared by
Rakesh Vidyadharan

Prepared for
Interested Developers/Deployers

Dated: October 30, 2007

Contents

1	Introduction	3
1.1	Features	3
1.2	Technology	4
2	Installation	4
2.1	Authentication	5
3	Customisation	5
3.1	Building Application	5
3.2	Localisation	6
3.3	Look-And-Feel	6
3.3.1	Using DataSource	6
3.3.2	Using Other Databases	7
A	FAQ's	7
A.1	How do I specify statements in the Batch Query Executor?	7
A.2	Why do not paginate batch query results?	8
A.3	How do you implement pagination of results?	8
A.4	Why do you not show full meta data for Oracle/Sybase?	8
B	Use Cases	8
B.1	Connection Use Cases	8
B.2	Database Objects	10
B.3	Query Tools Use Cases	13
B.3.1	Query Executor Use Cases	16
B.3.2	Batch Query Executor Use Cases	18

List of Tables

1	Software Requirements	5
2	DataSource Use Case	9
3	Connection Use Case	9
4	Save Connection Use Case	10
5	Database Objects Use Case	11
6	Table Details Use Case	12
7	Procedure Details Use Case	13
8	Trigger Details Use Case	13
9	Statement History Use Case	14
10	Save Query Use Case	15
11	Query Executor Use Case	16
12	Execute Query Use Case	17
13	Batch Query Executor Use Case	18

1 Introduction

The SPT Relational Database Web Tool (RWT) is a simple application that can help developers connect to databases configured as [DataSource](#) in the application server, or directly through JDBC. The RWT application attempts to replicate basic features found in the popular Aqua Data Studio application. Some simple use cases used as the primary requirements for developing the tool are listed in the appendix.

1.1 Features

The following are the basic features provided by the tool:

1. Connect to databases configured in application server as [DataSources](#). Configured [DataSources](#) are displayed in the application menu.
2. Display a [Connection](#) dialogue through which users may specify the parameters for directly connecting to a database through JDBC. The connection parameters may be saved in which case the [Connection](#) will be added to the application menu of available databases.
3. Display in a [Tree](#) the following types of database objects available in the connected database.
 - 3.1. The schemas accessible to the user.
 - 3.2. The tables available under a schema.
 - 3.3. The views available under a schema.
 - 3.4. The procedures and functions available under a schema.
 - 3.5. The triggers defined in a schema.
 - 3.6. The constraints (primary and foreign keys) defined in a schema.
4. A query execution tool launched through a menu (multiple windows are supported). The query tool makes the following features available:
 - 4.1. A free text area in which the user enters the query they wish to execute.
 - 4.2. An optional text field which can be used to limit the maximum number of matching records fetched from the database.
 - 4.3. A “Save” button that can be used to save the current query for easy access via a menu in the future.
 - 4.4. The results of the query are displayed in tabular format. The results are pageable and sortable for convenience.
 - 4.5. The results of the query can be exported to Excel through a “Export to Excel” button.

1.2 Technology

The RWT application was developed using [Echo2](#) and [EchoPointNG](#) for the User Interface and JDBC for database access. Database metadata is fetched using standard JDBC as well as the Information Schema supported by most databases.

[Echo2](#) was used to develop the User Interface to maximise developer productivity. [Echo2](#) helps developers avoid the impedance mismatch between having to implement business logic in [Java](#), and implement the display logic using totally different languages such as [XHTML 1.0](#), [JavaScript](#), ...

2 Installation

The RWT application is relatively easy to install. Table 1 shows the minimum versions of the software necessary to install the application. The following steps need to be executed to deploy the application to the application server.

1. Download the `sptrwt.zip` file from the project home [page](#).
2. Extract the `sptrwt.war` file from the downloaded zip archive.
3. Create the central directory under which application support files are located. Support files are used to store saved [Connection](#) details, saved queries etc. Note that the central directory may need to be on a file server if you wish to deploy the application to multiple nodes. For example, [Sans Pareil Technologies, Inc.](#) uses a `/var/data/rwt` directory as the application directory.
4. Configure a [DataSource](#) named `jdbc/rwt` in your application server. Refer to the building instructions (see section 3.1) if you wish to configure additional [DataSources](#), or use a different [DataSource](#) name.
5. Configure a JVM system property named `sptrwt.data.directory` for the container. Listing 1 shows the steps involved in configuring the property. For [Tomcat](#), example, add a line similar to the following to `$CATALINA_BASE/bin/catalina.sh`
6. Deploy the `war` file to the application server.

Listing 1: JVM Property Configuration

```
$ cd $CATALINA_HOME/bin
$ echo "JAVA_OPTS=\"$JAVA_OPTS -Dsptrwt.data.directory=/var/data/rwt\" \" \" \
>> setenv.sh
```

Table 1: Software Requirements

Software	Version	Description
Java	1.5	Minimum version of the JRE that is required
Servlet	1.3	Minimum version of the Servlet API that is required.
PostgreSQL	8.2	Version of the database engine application was tested with.
Oracle	10.2.0.2.0	Version of the database engine application was tested with.
MySQL	5.0.41	Version of the database engine application was tested with.
Sybase	12.50	Version of the database engine application was tested with.
Tomcat	5.5.17, 6.0.14	Versions application was tested with.

2.1 Authentication

It is possible to configure container managed authentication for the application. User preferences such as saved connections and saved queries will be stored in a directory named after their `username` under the application `datadirectory`. The `web.xml` file included with the application contains a commented out authentication configuration section that can be enabled and modified to match your environment.

3 Customisation

The application will need to be customised to suit different requirements. Customisation includes changing the look-and-feel, localising the text on screen and also the application database. Customising the application requires rebuilding the application from the sources. Section 3.1 describes the steps required to build the application from the sources available in the source repository.

3.1 Building Application

The application source code may be checked out from the source repository and built locally. This will be necessary if you wish to customise any aspect of the application. This will also be necessary if you wish to make changes to the application source code to meet your requirements. Listing 2 shows the steps involved in checking out and building the application. See sections 3.3.1 and 3.3.2 for details on how to configure the database server and application server for deployment.

Listing 2: Building Application

```
$ svn co https://sptrwt.dev.java.net/svn/sptrwt/trunk \
  sptrwt --username <username>
$ cd sptrwt
$ cp ant.properties.sample ant.properties
```

```
$ vi ant.properties # Edit location of dir.appserver
$ vi config/context.xml # Applies to Tomcat
# Edit jdbc/rwt database as appropriate or delete
# Add additional datasources as appropriate
$ ant test
# Will run a sequence of JUnit tests to test the core database
# abstraction API used by the application.
$ ant # Will compile and deploy application to dir.appserver
$ ant clean # Will clean up all binaries created in previous step
```

3.2 Localisation

Most of the labels and other static text displayed in the RWT application are configured for each component in a `properties`¹ file. The property values may be modified to suit different requirements. Copies of the file may be created for other languages, which will be loaded based upon client browser language preference using regular [Java](#) property file loading rules.

3.3 Look-And-Feel

Look-and-feel of the application are controlled through a set of [Style](#)² classes in the `style`³ packages. You may edit these files as appropriate to modify the look-and-feel to suit your requirements.

3.3.1 Using DataSource

In order to modify or add [DataSources](#) some modifications are necessary to the standard `context.xml` file that is packaged with the application⁴.

Listing 3: Using DataSource

```
$ vi config/context.xml
# Only if you are deploying to Tomcat or if your application server
# supports META-INF/context.xml in your war file.
# Edit the <Resource name='' username='' etc.~as appropriate
# Add additional <Resource name='' elements as appropriate.
$ vi config/web.xml
# Modify the <resource-ref> configuration at the end of the file.
# Edit the <res-ref-name> element as appropriate
# Specify additional datasources using additional resource-ref elements.
$ ant war
```

¹config/resource/localisation/Configuration.properties

²<http://docs.rakeshv.org/java/echo2/app/nextapp/echo2/app/Style.html>

³`com.sptci.echo2.style` and `com.sptci.epng.style`

⁴This works for Tomcat. Consult your application server documentation to determine the steps required to configure your application server.

```
# You can instead run ant if deploying to local machine
```

3.3.2 Using Other Databases

You can connect to any database that provides a JDBC driver and supports the Information Schema. Listing 4 shows the steps that you would need to perform to configure another database engine (Oracle in this example) with the application.

Listing 4: Using Other Database

```
$ cp <path to jdbc driver> $CATALINA_HOME/lib # For Tomcat 6
$ cp <path to jdbc driver> $CATALINA_HOME/common/lib # For Tomcat 5
$ vi config/context.xml # Edit properties similar to following
#   driverClassName='oracle.jdbc.driver.OracleDriver'
#   url='jdbc:oracle:thin:@sunU60:1521:spt'
$ vi ant.properties # Edit database connection information
$ ant test # Run unit tests against new database.
$ ant war
# You can instead run ant if deploying to local machine
```

A FAQ's

Sections A.x list a few frequently asked questions about the application.

A.1 How do I specify statements in the Batch Query Executor?

The batch query executor uses some very simple rules to parse and tokenise the statements entered. The parser logic is the same as that listed in the [PostgreSQL Forum](#)⁵. Some databases such as [PostgreSQL](#), [Sybase](#) (and probably MS SQL Server) supports directly sending a batch of SQL statements without any client side parsing. The statements you enter are sent as is to these databases. For Sybase (and probably MS SQL Server), you will need to delimit the queries by the `go` keyword.

For other databases, the parser uses the Oracle standard `/` character on a sepearate line as the delimiter between statements. In addition you will need to terminate each statements with a semi-colon. This approach naturally restricts the type of statements you can enter into the query executor. You can enter complex statements (create complex functions/procedures, tables, views etc) all in one big batch (or upload from a script file) for [PostgreSQL](#). You may be able to do something similar with Sybase (we have not tested extensively against Sybase due to lack of test facilities).

⁵<http://archives.postgresql.org/pgsql-jdbc/2006-02/msg00034.php>

A.2 Why do not paginate batch query results?

The batch query executor attempts to execute the batch of SQL statements as a single statement⁶ which makes it impossible to paginate the results of the individual statements that comprise the batch that was input.

A.3 How do you implement pagination of results?

Pagination of results in the Query Executor is accomplished by re-executing the statement to fetch each page of data. Only the rows necessary to populate a page is retrieved on each statement execution. This helps to reduce the server resources needed to maintain the application at the cost of multiple requests to the database server.

A.4 Why do you not show full meta data for Oracle/Sybase?

Details that are not available through JDBC are retrieved from the `information_schema`. Neither Oracle nor Sybase implement these SQL standard views. In a future release more support for querying the Oracle data dictionary to fetch additional details will be added.

B Use Cases

The following sections describe some simple use cases that were used to define the requirements for the application.

B.1 Connection Use Cases

Table 2 defines the process of connecting to a [DataSource](#) configured for the application. Table 3 defines the process of initiating a direct [Connection](#) to a database. Table 4 defines the process of saving a connection to the application menu for future use.

⁶When supported by the database. [PostgreSQL](#) and Sybase support this (and possibly MS SQL Server). Native batch execution allows you to specify complex SQL statements that would be difficult to reliably parse without a full fledged SQL statement parser. Executing as a single statement however does not provide access to the individual statements that comprise the batch, thus preventing database pagination of the results. Client side pagination would be possible by using a [PageableSortableTable](#), but it is not designed to accomodate anything more than a couple of hundred records just defeatin the purpose of paginating the results.

Table 2: DataSource Use Case

Use Case	1
Use Case Name	Configured DataSource
Summary	Connecting to DataSources configured for the application.
Description	The “Data Sources” menu item under “Server” menu displays all the configured DataSources for the RWT application.
Post-condition	The selected DataSource will be used to obtain JDBC Connections for the rest of the session, or until another DataSource or Connection is selected from the “Data Sources” menu item or “Connections” menu item.

Table 3: Connection Use Case

Use Case	2
Use Case Name	Connection via JDBC
Summary	Connecting to a database directly through JDBC.
Description	<ul style="list-style-type: none">• Select the “Connections” menu item under “Server” menu to display the dialogue for specifying the Connection parameters.• Save the connection parameters for future use. See table 4 for details.• Click the “Connect” button to connect to the specified database.
Post-conditions	The specified Connection will be used for the rest of the session, or until another DataSource or Connection is selected from the “Data Sources” menu item or “Connections” menu item.

Table 4: Save [Connection](#) Use Case

Use Case	3
Use Case Name	Save Connection .
Summary	Save a JDBC Connection for future use to the application menu.
Pre-condition	Launch the connection dialogue as described in table 3.
Description	<ul style="list-style-type: none">• Click the “Save” button to save the Connection parameters to the “Connections” menu item under “Server” menu.• A save connection dialogue is displayed that can be used to specify the unique name to assign for the connection.
Post-condition	<ul style="list-style-type: none">• The “Connections” menu item under “Server” menu is updated with the newly saved connection.• The configuration file in which saved connections are stored is updated to save the information for future sessions. Note that the database user’s password is stored in the configuration file. The password value is stored encrypted using AES 128 bit encryption, and hence should be reasonably secure.

B.2 Database Objects

The use cases in this section describe the various database objects that are displayed in the [Tree](#) displayed on the left side of the RWT application.

Table 5: Database Objects Use Case

Use Case	4
Use Case Name	Database objects.
Summary	The database objects that are displayed in the Tree in the application.
Pre-condition	Select a database to connect to as described in the use cases in section B.1 .
Description	<p>Database objects are displed are tree nodes under a parent Schema node in the Tree.</p> <p>Tables A node that is the parent of child nodes that represent all the tables available in the schema.</p> <p>Views A node that is the parent of child nodes that represent all the views available in the schema.</p> <p>Procedures A node that is the parent of child nodes that represent all the procedures and functions available in the schema.</p> <p>Triggers A node that is the parent of child nodes that represent all the triggers configured in the schema.</p> <p>Constraints A node that is the parent of child nodes that represent all the constraints configured in the schema.</p>

Table 6: Table Details Use Case

Use Case	5
Use Case Name	Table details.
Summary	The details about table objects that are displayed by the application.
Pre-condition	Select a database to connect to as described in the use cases in section B.1 .
Description	<p>The following child nodes are displayed for each table displayed in the Tree.</p> <p>Columns Displays a node under which all the columns defined for the table are displayed.</p> <ol style="list-style-type: none"> 1. Clicking on the node displays all the columns and their associated metadata in tabular format adjacent to the Tree. 2. Clicking on the child nodes that represent each column displays the metadata for the column adjacent to the Tree. <p>Constraints Displays a node under which all the constraints defined for the table are displayed.</p> <ol style="list-style-type: none"> 1. Clicking on the node displays all the constraints and their associated metadata in tabular format adjacent to the Tree. 2. Clicking on the child nodes that represent each constraint displays the metadata for the constraint adjacent to the Tree. <p>Indices Displays a node under which all the indices defined for the table are displayed.</p> <ol style="list-style-type: none"> 1. Clicking on the node displays all the indices and their associated metadata in tabular format adjacent to the Tree. 2. Clicking on the child nodes that represent each index displays the metadata for the index adjacent to the Tree. <p>Triggers Displays a node under which all the triggers defined for the table are displayed.</p> <ol style="list-style-type: none"> 1. Clicking on the node displays all the triggers and their associated metadata in tabular format adjacent to the Tree. 2. Clicking on the child nodes that represent each triggers displays the metadata for the triggers adjacent to the Tree.

Table 7: Procedure Details Use Case

Use Case	6
Use Case Name	Procedure objects.
Summary	The details about procedure/function objects that are displayed by the application.
Pre-condition	Select a database to connect to as described in the use cases in section B.1 .
Description	The details of a selected procedure/function are displayed in tabular format adjacent to the Tree . The details are retrieved from the Information Schema .

Table 8: Trigger Details Use Case

Use Case	7
Use Case Name	Procedure objects.
Summary	The details about trigger objects that are displayed by the application.
Pre-condition	Select a database to connect to as described in the use cases in section B.1 .
Description	The details of a selected trigger are displayed in tabular format adjacent to the Tree . The details are retrieved from the Information Schema .

B.3 Query Tools Use Cases

The following use cases define the features that are supported by the query tools and its associated menu.

Table 9: Statement History Use Case

Use Case	8
Use Case Name	Statement History.
Summary	SQL statements executed by the use for a query executor view component is saved and viewable.
Pre-condition	<ol style="list-style-type: none">1. Select a database to connect to as described in the use cases in section B.1.2. Launch the query tool as described in table 11 or 13.
Description	<ul style="list-style-type: none">• Input the SQL statement you wish into the query area in the query tool.• Click the “Execute” button to execute the SQL statement.• The statement is automatically added to the history for the view.• Click the “History” button to launch the Statement History view.• Click on any of the previously executed statements^a to set the selected statement in the executor query area.

^aThe view displays only up to the first 100 characters in the statement.

Table 10: Save Query Use Case

Use Case	9
Use Case Name	Save Query.
Summary	Save query entered by user into query tool text area.
Pre-condition	<ol style="list-style-type: none">1. Select a database to connect to as described in the use cases in section B.1.2. Launch the query tool as described in table 11.
Description	<ul style="list-style-type: none">• Input the SQL statement you wish into the query area in the query tool.• Click the “Save” button to save the SQL statement.• Enter a unique name for the query and a category under which to file the query in the save query dialogue.
Exception	If the name specified by the user is not unique, an error message is displayed to the user, and prompted to enter another name.
Post-condition	The SQL statement with the specified name becomes available under the specified category under the “Query” menu.

B.3.1 Query Executor Use Cases

Table 11: Query Executor Use Case

Use Case	10
Use Case Name	Query Executor.
Summary	Basic features of the query executor provided by the application.
Pre-condition	Select a database to connect to as described in the use cases in section B.1 .
Description	<p>The query tool can be launched by selecting the “Query Executor” menu item under “Query Tools” menu. The tool provides the following features:</p> <ul style="list-style-type: none"> • A TextArea into which the SQL statement can be entered by the user. • A TextField that can be used to restrict the maximum number of records to be retrieved from the database upon query execution. • A “Execute” button that can be used to execute the entered query against the selected database. The results obtained are displayed in a Pageable Table. Paging is implemented by fetching the total number of records for the result set and calculating the number of pages available. Page navigation results in the SQL statement being executed again to fetch the data for the page requested. This helps keep client side memory requirements to a minimum. • A “Export” button that can be used to export the entire result set associated with the SQL statement entered as an Excel workbook. • A “Save” button that can be used to save the entered query for future use.

Table 12: Execute Query Use Case

Use Case	11
Use Case Name	Execute Query.
Summary	Execute query entered by user against the selected database.
Pre-condition	<ol style="list-style-type: none">1. Select a database to connect to as described in the use cases in section B.1.2. Launch the query tool as described in table 11.
Description	<ul style="list-style-type: none">• Input the SQL statement you wish into the query area in the query tool.• Click the “Execute” button to execute the statement against the database.
Exception	An error window is displayed to the user with the details about the cause of the error if the statement execution failed.
Post-condition	<p>The results of query execution are displayed in tabular format.</p> <ol style="list-style-type: none">1. Records retrieved from the database are displayed in a pageable table.2. The results of the query execution can be exported as an Excel file by clicking on the “Export to Excel” button.

B.3.2 Batch Query Executor Use Cases

Table 13: Batch Query Executor Use Case

Use Case	12
Use Case Name	Batch Query Executor.
Summary	Basic features of the query executor provided by the application.
Pre-condition	Select a database to connect to as described in the use cases in section B.1.
Description	<p>The query tool can be launched by selecting the “Batch Query Executor” menu item under “Query Tools” menu. The tool provides the following features:</p> <ul style="list-style-type: none"> • A TextArea into which the SQL statement(s) can be entered by the user. • A TextField that can be used to restrict the maximum number of records to be retrieved from the database for each statement entered. • A “Execute” button that can used to execute the entered query against the selected database. The results obtained for each statement is displayed in a separate tab in a Tabbed Pane. • A “Upload” button that can be used to upload a script file containing the SQL statements that are to be executed in one batch. • A “Save” button that can be used to save the entered query for future use.