



EXTENSIBLE
CATALOG

Unifying access to library resources

NCIP Developer Documentation

Version 0.5.3

NCIP Toolkit: Developer Documentation

© eXtensible Catalog
University of Rochester
1325 Mt. Hope, Suite 102
PO Box 278960
Rochester, NY 14627-8960
<http://www.xcproject.org/>

Table of Contents

Introduction	3
Request Listener	3
Request Handler	3
ILS Interface	4
Adding support for a new ILS.....	5
The NCIP Toolkit's interpretation of NCIP.....	5
Adding configurable parameters	6
NCIP Objects.....	7
NCIP Agency	10
NCIP Item	10
NCIP User	15
NCIP User Fine.....	16
NCIP User Loaned Item	16
NCIP User Requested Item.....	16
XC User Recalled Item	17
XC Item Availability	17
Error Reporting	17
Error Codes	17
Internal errors (errors not caused by incoming requests or errors in the NCIP Toolkit configuration file)	17
Generic NCIP Request errors	18
Configuration file errors.....	18
Authentication errors.....	18
Request Item Service errors.....	18
Cancel Request Item Service errors	18
Recall Item Service errors	19
Cancel Recall Item Service errors.....	19
Renew Item Service errors.....	19
The StrictNcipError Object	19
ILSInterface methods	20

Introduction

This document is intended to provide guidelines for developers who are modifying the NCIP Toolkit. It begins by outlining the design of the NCIP Toolkit, and then provides instructions for adding support for a new ILS to the NCIP Toolkit. The reader is expected to have a basic understanding of Java and Java Servlets. They are also expected to have read the NCIP Toolkit User Documentation and have a strong understanding of the NCIP Toolkit from a user perspective, including knowing how to send requests and interpret responses. The NCIP Toolkit User Documentation can be downloaded at <http://docushare.lib.rochester.edu/docushare/dsweb/Get/Document-31339/NCIPToolkitUserDocumentation.docx>.

The NCIP Toolkit consists of two parts, the Core and a driver implementation.

The core contains the following two components:

- 1) Request Listener – Servlet which accepts and handles NCIP requests
- 2) Request Handler – Parses request data, calls the ILS Interface, and returns an NCIP response

The driver implementation contains an ILS Interface that executes requests against an ILS

Each of these components is explained below.

Request Listener

The Request Listener component is run as a Servlet on a system running the NCIP Toolkit. It accepts a request and runs the Request Handler component in the Thread handling the connection which made the request. It will handle HTTP and HTTPS POST requests with the NCIP request stored in a POST data parameter called “ncip”. When a request is made to the Request Listener, it will return the response and keep the connection alive for a number of seconds defined in a configuration file common to all components of the NCIP Toolkit.

The Request Handler Object which runs NCIP requests is preserved in the session state for a connection under the key “RequestHandler”. This allows the NCIP Toolkit to track authentication data for a user who has logged into the ILS. If the ILS times out a user’s login before the NCIP Toolkit’s timeout expires, the NCIP Toolkit will be able to re-authenticate the user without asking for their login information a second time. Alternatively, the ILSInterface may start a “heartbeat Thread” to make periodic inconsequential requests of the underlying ILS to ensure that the ILS never times out the user (this is the approach used for interacting with Voyager.)

Request Handler

The Request Handler component is invoked by the Request Listener when a request is received. The RequestHandler class acts as the entry point into the Request Handler component. Each client’s requests are handled by a different instance of a RequestHandler Object stored in the session specific to

NCIP Toolkit Developer Documentation

the connection to the Request Listener Servlet. A RequestHandler Object contains an ILSInterface Object (described later in this document) to run requests on the ILS. This ILSInterface maintains any data from previous requests which may be needed by future requests (such as authentication data.)

An abstract class called NCIPRequest with a two protected abstract methods, parse and execute, assists with handling incoming NCIP Requests. The parse method requires no parameters and is responsible for parsing the necessary data from the request. The XML for the NCIP Request is stored in a protected Document Object called ncipReqXML before the parse method is called. The execute method takes no parameters and invokes the correct functionality on the ILS Interface component. The execute method returns the NCIP response as a string based on the result from the ILS Interface.

The NCIPRequest class contains a public executeRequest method which will setup the ILS Interface based on values defined in the driver configuration file, set the ncipReqXML Document Object to the incoming NCIP Request, call the parse method, then call the execute methods. This allows subclasses to assume that the NCIP Request will be initialized before parse is called and that parse will always be called before execute.

For each supported NCIP command, there is a class which extends the NCIPRequest abstract class. The RequestHandler parses the NCIP request into a Document Object and runs an XPATH expression to find out the type of request it is handling. It then creates an instance of the appropriate Object extending the NCIPRequest abstract class, tells this Object which ILS Interface to perform calls on, and calls its executeRequest method passing a reference to the Document Object. Assuming there was not a parse error, it will then call the NCIPRequest's execute method and send the response returned across the connection. It will then continue to listen on the connection for further NCIP requests to handle. It will close the connection when the client closes their connection or after a timeout expires whose duration is read from the NCIP Toolkit configuration file.

The RequestHandler handles the following NCIP requests: Authenticate User, Lookup Item, Lookup User, Lookup Version, Recall Item, Cancel Recall Item, Request Item, Cancel Request Item, and Renew Item. Details on each request can be found in the NCIP Toolkit User Documentation. It also handles two requests which are not part of the NCIP protocol, XC Lookup User and XC Get Availability, which are describe in the NCIP Toolkit User Documentation.

ILS Interface

The ILS Interface component takes the details from a parsed NCIP request and executes the request on the ILS it is configured to run against by interacting with the ILS's database or OPAC depending on how the class for interfacing with that ILS is implemented. An ILSInterface interface will be defined with the methods necessary to run all supported NCIP requests. For each ILS the NCIP Toolkit can run against there is a class which implements ILSInterface and defines the methods to run against the specific ILS. For example, the VoyagerInterface class defines these methods to execute on a Voyager system

NCIP Toolkit Developer Documentation

RequestHandler Objects create an instance of the appropriate class implementing ILSInterface when they are constructed. The driver configuration file will contain the type of ILS the RequestHandler will interface with in the parameter ILSType. It also contains the InterfaceName and the DriverFile which are the interface name and the jar file respectively. This allows the RequestHandler to know which implementer of ILSInterface to use. This configuration file also contains any other information necessary to connect with the ILS, such as the URL at which it can be accessed.

Adding support for a new ILS

In order to add support for a new ILS, you must first create a new class package which implements the ILSInterface class. The methods you must implement are explained in detail in the “ILSInterface methods” section of this document.

A configuration file called driver_config.xml should be created to store each interfaces properties.

Example project structure:

```
| -src
    | -xc.nciptoolkit.<ils name>.java
    | -xc.nciptoolkit.constants.Constants.java (Constants file extending
    xc.nciptoolkit.constants.Constants)
| -build
    | -classes (compiled .class files)
    | -dist
        | -NCIPToolkit-<ils name> (Created by Ant)
        | -NCIPToolkit-<ils name>.zip (Created by Ant)
| -conf
    | -driver_config.xml
| -lib
    | -NCIPToolkit-Core.jar (compiled jar file from core used for compiling)
    | -Any other required .jar files
| -build.properties (Ant build properties)
| -build.xml (Ant build file)
```

Hereafter, the phrase “ILSInterface class” will refer to a class implementing the ILSInterface interface, and the phrase “ILSInterface Object” will refer to a specific instance of an ILSInterface class.

The NCIP Toolkit’s interpretation of NCIP

The NCIP Toolkit supports the following NCIP Services: Authenticate User, Lookup Item, Lookup User, Lookup Version, Request Item, Recall Item, Renew Item, Cancel Request Item, and Cancel Recall Item. Requests for any other NCIP Services will result in an Unsupported Service error being reported. For the supported NCIP Services, every effort was made to make the NCIP Toolkit perfectly NCIP compliant, but there were two areas where the NCIP Toolkit varies from the standard.

NCIP Toolkit Developer Documentation

The NCIP protocol states that NCIP is a sessionless protocol. However, most ILSs require a user to authenticate before permitting them to submit request, recall, or renew requests. For this reason the NCIP Toolkit does maintain a session for each connection with information on the user it most recently authenticated. When a Request Item, Recall Item, Renew Item, Cancel Request Item, or Cancel Recall Item request is made which does not provide Authentication Inputs, the NCIP Toolkit will place the request for the user authenticated in the session, except when the request specifies a Unique User Id which differs from the authenticated user in which case it returns an Authentication Required error.

Since the NCIP Request Item, Renew Item and Cancel Request Item requests allow either a Unique User Id or Authentication Inputs, the decision to maintain this session information for connections to the NCIP Toolkit does not impact a client connecting to it expecting a strict interpretation of NCIP. If such a client provides a Unique User Id with these requests, the NCIP Toolkit will respond with an Authentication Required error. The client with the strict interpretation would then know to provide Authentication Inputs for these requests. However, the Recall Item and Cancel Recall Item requests do not allow any means to specify the user who is placing the recall. If an ILS requires authentication to place or cancel recalls, the client connecting to the NCIP Toolkit must place an Authenticate User service for the user placing the request before placing a Recall Item or Cancel Recall Item request.

The NCIP Toolkit can also be configured to report errors according to a strict or loose interpretation of the NCIP protocol. If it is configured to use strict error reporting it will return Problem elements which adhere to the NCIP standard. However, if configured to use loose error reporting, the NCIP Toolkit will report errors using Problem elements that differ from the structure the NCIP standard defines for this element. More information on this difference can be found in the NCIP Toolkit User Documentation in the NCIP Toolkit Problem Element section. Information on how error reporting should be implemented in new ILS Interface classes is in the Error Codes section below.

Adding configurable parameters

While implementing these methods, it is likely that you will need to expose parameters in the NCIP Toolkit driver configuration file so they may be configured for a specific install of the ILS (for example, the URL to access the ILS.) To do so, add the following line to the <category name="general"> element of the file driver_config.xml:

```
<property name="<ILS_name><parameter_name>" value="<default_value>" />
```

Where ILS_name is the name of the ILS the parameter is used for (Voyager, Aleph, Evergreen, etc.,) parameter_name is the name of the configurable parameter, and default_value is the value this parameter should have if it has not been configured.

You must then add the following line to the constants file located in your driver implementation:

```
public static final String CONFIG_<formatted_full_parameter_name> = "<full_parameter_name>";
```

NCIP Toolkit Developer Documentation

Where `full_parameter_name` is the name of the key (name attribute of the property) added to the configuration file, and `formatted_full_parameter_name` is the full parameter name with only capital letters and words separated by underscores.

At the beginning of the implementation of `ILSInterface` class you're writing, you must import the following package:

```
xc.nciptoolkit.utils.ILSConfiguration;
```

The following code fragment can then be used to pull the value of a configurable parameter from the configuration file as a String:

```
ILSConfiguration.getProperty( <constantsfile>.CONFIG_<formatted_full_parameter_name>)
```

NCIP Objects

The NCIP Toolkit uses several classes stored in the `ncipObjects` project to pass data between the ILS Interface and Request Handler components, and to assist with building the response to the NCIP request. The .java files for these classes should not need to be modified, but it is important to understand how they work since several of the `ILSInterface` methods accept NCIP Objects as parameters and are required to build and return them.

All of the NCIP Objects have required and optional fields. The required fields are accepted by the constructor of an NCIP Object class, and the class contains getter and setter methods for all optional and required fields. The required fields are needed to build a valid NCIP Response, and the optional fields will be included in the NCIP Response if and only if they were set by calling the corresponding setter on the NCIP Object.

Similarly to NCIP Objects, the NCIP Toolkit also makes use of XC Objects. XC Objects work the same way as NCIP Objects, but they are used to add functionality which goes beyond the NCIP protocol.

Some NCIP Objects and XC Objects have as optional fields an `ArrayList` of Strings or other NCIP Objects. For example, an `NCIPUser` may contain one or more `NCIPUserFine` Objects. In these cases, the getter method returns the entire `ArrayList`, and in place of the setter method there is an `add-to-list` and a `reset-list` method.

The NCIP Toolkit uses the following NCIP Objects and XC Objects:

NCIP Accept Item Response

Optional Fields:

- `NCIPRequestId` request – The ID of the request
- `NCIPItem` item – The item that was accepted

NCIP Bibliographic Description

Optional Fields:

NCIP Toolkit Developer Documentation

- String author – The author of the item
- String authorOfComponent – The author of a component of the item
- String bibliographicItemIdentifier – The bibliographic item identifier of the item
- String bibliographicItemIdentifierCode – A code for the type of the bibliographic identifier
- String bibliographicLevel – The bibliographic level the description describes
- String bibliographicRecordIdentifier – An identifier of the bibliographic record
- String bibliographicRecordIdentifierCode – A code for the type of record identifier
- NCIPAgency bibliographicRecordIdentifierAgency – The agency to which the record ID pertains
- String bibliographicComponentIdentifier – An identifier of the component
- String bibliographicComponentIdentifierType – The type of component identifier
- String edition – The edition described by the bibliographic entry
- String electronicDataFormatType – The electronic format of items described by the entry
- String language – The language described by the bibliographic entry
- String mediumType – The type of medium the items are expressed with
- String pagination – The pagination of items described by the bibliographic entry
- String placeOfPublication – Where the entry was published
- Date publicationDate – When the entry was published
- Date publicationDateOfComponent – When the entry's component was published
- String publisher – The publisher of the item
- String seriesTitleNumber – The number in a series of the item
- String sponsoringBody – The entity which sponsored the item
- String title – The title of the item
- String titleOfComponent – The title of a component of the item

NCIP Block Or Trap

Optional Fields:

- String blockOrTrapType – The type of block or trap which has been placed
- NCIPAgency agency – The agency that issued the block or trap
- Date validFromDate – The date from which the block or trap is valid
- Date validToDate – The date until which the block or trap is valid

NCIPCheckInItemResponse

Optional Fields:

- NCIPItem item – The item which was checked in
- NCIPUser user – The user who checked in the item
- NCIPRoutingInformation routingInformation – Routing information for returning the item
- NCIPFiscalTransactionInformation fiscalTransactionInformation – Information on fiscal transactions related to checking in the item

NCIPCheckOutItemResponse

Optional Fields:

NCIP Toolkit Developer Documentation

- Date dateDue – The date when the item is due
- boolean nonReturnableFlag – True if the item is non-returnable, false otherwise
- boolean indeterminateLoanPeriodFlag – True if the item's due date is not known, false otherwise
- NCIPItem item – The item to be checked out
- NCIPUser user – The user checking out the item

NCIPFiscalTransactionInformation

Optional Fields:

- String currencyCode – The code representing the type of currency in which the transaction's amount is represented according to ISO 4217 (ex: USD for US dollar)
- int amount – The amount of the transaction in the smaller value of the currency defined in currencyCode. For example, if currencyCode = USD and amount = 1000, then the user owes \$10.00.
- String actionType – The type of action taken for the fiscal transaction
- String description – A description of the fiscal transaction
- String identifier – An identifier of the fiscal transaction
- NCIPAgency agency – The agency which the transaction involved
- String transactionReferenceId – The ID of a transaction referenced by this transaction
- NCIPAgency transactionReferenceAgency – The agency which the referenced transaction involved
- String type – The type of fiscal transaction
- NCIPItemDetails details – Details about the item related to the fiscal transaction
- String paymentMethod – The method used to pay for the transaction
- NCIPRequestId requestId – The ID of the request which the transaction involved
- Date validFrom – The date from which the transaction is valid
- Date validTo – The date until which the transaction is valid

Optional Lists:

- List<NCIPFiscalTransactionReference> relatedTransactions – A list of IDs of transactions related to this transaction

NCIP Fiscal Transaction Reference

Optional Fields:

- NCIPAgency agency – The agency which the referenced transaction involved
- String transactionId – The ID of the referenced transaction

NCIP Holdings Chronology Level

Optional Fields:

- String caption – The descriptive term identifying the chronology (Day, Year, etc.)

NCIP Toolkit Developer Documentation

- int level – The level of the chronology
- String value – The value of the chronology

NCIP Holdings Enumeration Level

Optional Fields:

- String caption – The descriptive term identifying the enumeration (volume, part, etc.)
- int level – The level of the enumeration
- String value – The value of the enumeration

NCIP Agency

Required Fields:

- String agencyIdentifier – A String identifying the agency

NCIP Item

Required Fields:

- NCIPAgency holdingAgency – The agency which is circulating the item
- String itemId – The item's item ID

NOTE: An alternate constructor accepts an ID and an ID type. This constructor will set the correct ID (item, bib, or holdings) based on the ID type provided. If this constructor is used, the holdingAgency will be set to the default defined in the configuration file.

Optional Fields:

- String bibId – The item's bibliographic ID
- String holdingsId – The item's holdings ID
- String circulationStatus – A String explaining the item's circulation status (ex: "Checked Out")
- String electronicResource – A String identifying the item's electronic resource.
- int holdQueueLength – The number of patrons currently placing holds on the item
- String itemDescription – A String describing the item
- String location – The location of the item when it is not checked out
- String callNumber – The item's call number
- String author – The item's author
- String isbn – The item's ISBN identifier
- String mediumType – The type of medium which the item is (ex: "Book" or "DVD")
- String publisher – The item's publisher
- String series – The series to which the item belongs
- String title – The item's title

Optional Lists:

- List<NCIPUser> borrowingUsers – A list of users who have the item checked out

NCIP Toolkit Developer Documentation

- List<NCIPUser> requestingUsers – A list of users who have placed requests for the item

NCIP Item Description

Optional Fields:

- String callNumber – The item’s call number
- String copyNumber – The copy number of the item
- String unstructuredHoldingsInformation – Holdings information about the item
- String itemDescriptionLevel – The level described by the item description
- String visibleItemIdentifier – A visible (non-unique) identifier of the item
- String visibleItemIdentifierType – The type of the visible item identifier (ex: “Bibliographic ID”)

Optional Lists:

- List<NCIPHoldingsChronologyLevel> – Contains all the chronology information about the item’s holdings
- List<NCIPHoldingsEnumerationLevel> – Contains all the enumeration information about the item’s holdings

NCIP Item Details

Optional Fields:

- NCIPBibliographicDescription bibInfo – Bibliographic information on the item
- Date dateCheckedOut – The date when the item was checked out, if applicable
- Date dateDue – The date when the item is due for return, if applicable
- Date dateReturned – The date when the item was returned, if applicable
- NCIPItem item – The item being described
- boolean nonReturnable – True if the item is not returnable, false otherwise
- boolean indeterminateLoanPeriod – True if the item’s due date can’t be determined, false otherwise

Optional Lists:

- List<Date> datesRenewed – The dates when the item was renewed, if applicable

NCIP Item Optional Fields

Optional Fields:

- NCIPBibliographicDescription bibDescription – Bibliographic information about the item
- String circulationStatus – The item’s circulation status
- NCIPItemDescription – Item level information about the item
- Date mandatedAction – The date when an action on the item was carried out on the client
- String physicalCondition – The item’s physical condition
- String physicalConditionType – The type of condition described by the physicalCondition field
- String securityMarker – The security marker used to sensitize the item

NCIP Toolkit Developer Documentation

- boolean sensitizationFlag – True if the item has been sensitized, false otherwise
- NCIPItem itemId – The item ID of the item
- NCIPRequestId requestId – The ID of the request pertaining to the item
- String electronicResource – The item’s electronic resource
- String electronicResourceFormat – The format of the item’s electronic resource
- String referenceToElectronicResource – A reference (such as a URL) to the item’s electronic resource
- int holdQueueLength – The number of users who have requests for the item that are unfulfilled

Optional Lists:

- List<String> itemUseRestrictionTypes – Restrictions on the item’s use
- List<NCIPLocation> locations – Locations where the item is stored
- List<String> languages – Languages in which the item can be found

NCIPLocation

OptionalFields:

- String type – The type of location (permanent, temporary, etc.)
- Date validFrom – The date from which the location is valid
- Date validTo – The date until which the location is valid

Optional Lists:

- List<NCIPLocationName> name – The name of the item, structured according to the NCIP Protocol

NCIP Location Name

Optional Fields:

- String name – The name of the location
- int level – The level of the location. For example, an item stored on the 4th floor of building X might have “building X” as its level 1 name and “4th floor” as its level 2 name.

NCIPNameInformation

OptionalLists:

- List<NCIPOrganizationNameInformation> – A list of organization names described by the name information
- List<NCIPPersonalNameInformation> – A list of personal names described by the name information

NCIPOrganizationNameInformation

Optional Fields:

- String organizationName – The name of the organization

NCIP Toolkit Developer Documentation

- String organizationNameType – The type of the organization’s name

NCIPPersonalNameInformation

Optional Fields:

- String givenName – The first name
- String initials – The initials
- String prefix – The prefix to the name
- String suffix – The suffix to the name, if applicable
- String surname – The last name
- String unstructuredName – The entire name. This may be used in place of the 5 fields above, but not in addition to them

NCIPRequestedInformation

Optional Fields

- boolean bibDescriptionRequested – True if and only if the bibliographic description was requested
- boolean circStatusRequested – True if and only if the circulation status was requested
- boolean electronicResourceRequested – True if and only if the electronic resource was requested
- boolean holdQueueLengthRequested – True if and only if the hold queue length was requested
- boolean itemDescriptionRequested – True if and only if the item description was requested
- boolean locationRequested – True if and only if the item’s location was requested
- boolean physicalConditionRequested – True if and only if the item’s physical condition was requested
- boolean securityMarkerRequested – True if and only if the item’s security marker was requested
- boolean sensitizationFlagRequested – True if and only if the item’s sensitization flag was requested
- boolean authenticationInputRequested – True if and only if the user’s authentication input was requested
- boolean blockOrTrapRequested – True if and only if the user’s blocks and traps were requested
- boolean dateOfBirthRequested – True if and only if the user’s date of birth was requested
- boolean nameInformationRequested – True if and only if the user’s name information was requested
- boolean userAddressInformationRequested – True if and only if the user’s address was requested
- boolean userLanguageRequested – True if and only if the user’s language was requested
- boolean userPrivilegeRequested – True if and only if the user’s privileges were requested
- boolean visibleUserIdRequested – True if and only if the user’s visible ID was requested

NCIPRequestId

Required Fields:

NCIP Toolkit Developer Documentation

- requestId – The request’s ID
- requestsAgency – The agency with which the request is associated

NCIPRoutingInformation

OptionalFields:

- String binNumber – The bin number the item belongs in. Either the bin number or the location is required
- String location – The location where the item belongs
- String requestType – The type of request requiring the item to be routed
- NCIPUser user – The user who placed the request requiring the item to be routed

NCIP User Address Information

Optional Fields:

- String electronicAddress – The user’s electronic address
- String electronicAddressType – The type of electronic address defined (ex: “email”)
- String physicalAddressType – The type of physical address used
- String careOf – The care of part of the address
- String country – The country of the address
- String district – The district of the address
- String houseName – The name of the house
- String line1 – The first line of the address
- String line2 – The second line of the address
- String locality – The locality of the address
- String locationWithinBuilding – The location within the building described by the address
- String postOfficeBox – The P.O. Box of the address
- String postalCode – The postal code of the address
- String region – The region of the address
- String street – The street of the address
- String unstructuredAddressData – The entire address. This may be used in place of the above fields
- String unstructuredAddressType – The type of address described by the unstructured address data
- Date validFrom – The date from which the address is valid
- Date validTo – The date until which the address is valid

NCIPUserOptionalFields

Optional Fields:

- Date dateOfBirth – The user’s date of birth
- NCIPNameInformation – The user’s name

Optional Lists:

NCIP Toolkit Developer Documentation

- List<NCIPBlockOrTrap> blocksOrTraps – Blocks or traps on the user
- List<NCIPUserAddressInformation> addressInformation – A list of addresses for the user
- List<String> languages – A list of languages for the user
- List<NCIPUserPrivilege> privileges – A list of privileges the user has
- List<NCIPVisibleUserId> visibleIds – A list of visible IDs for the user

NCIPUserPrivilege

Optional Fields:

- String type – The type of privilege the user has
- NCIPAgency agency – The agency with which the user has the privilege
- String description – A description of the privilege
- String currencyCode – The code representing the type of currency in which the privilege's feeAmount is represented according to ISO 4217 (ex: USD for US dollar)
- int feeAmount – The cost of the privilege in the smaller value of the currency defined in currencyCode. For example, if currencyCode = USD and feeAmount = 1000, then the privilege cost \$10.00.
- String paymentMethodType – The method used to pay the fee for the privilege
- String privilegeStatusType – The status of the privilege
- Date privilegeStatusDate – The date when the status was applied to the privilege
- Date validFromDate – The date from which the privilege is valid
- Date validToDate – The date until which the privilege is valid

NCIP User

Required Fields:

- NCIPAgency usersAgency – The agency to which the user belongs
- String userId – A String uniquely identifying the user with respect to the usersAgency

Optional Fields:

- String fullName – The user's full name
- String address – The user's permanent address
- String tempAddress – The user's temporary address
- String emailAddress – The user's email address
- String totalFinesCurrencyCode – The code representing the type of currency in which the total fines are represented according to ISO 4217 (ex: USD for US dollar)
- int totalFines – The total fines the user owes in the smaller value of the currency defined in totalFinesCurrencyCode. For example, if totalFinesCurrencyCode = USD and totalFines = 1000, then the user owes \$10.00.

Optional Lists:

- ArrayList<String> blocks – A list of blocks which have been placed on the user

NCIP Toolkit Developer Documentation

- ArrayList<String> messages – A list of messages which apply to the user
- ArrayList<NCIPUserFine> fines – A list of fines which the user owes
- ArrayList<NCIPUserLoanedItems> checkedOutItems – A list of “loans” describing items checked out to the user
- ArrayList<NCIPUserRequestedItems> requestedItems – A list of items that the user has requested
- ArrayList<XCUserRecalledItems> recalledItems – A list of items that the user has recalled

NCIP User Fine

Required Fields:

- String currencyCode – The code representing the type of currency in which the amount is represented according to ISO 4217 (ex: USD for US dollar)
- int amount – The amount the user owes in the smaller value of the currency defined in currencyCode. For example, if currencyCode = USD and amount = 1000, then the fine is for \$10.00.
- String titleFineOwedFor – The title of the item the fine is owed for
- Date accrualDate – The date when the fine was incurred
- NCIPItem itemFinedFor – The item for which the user owes the fine

NCIP User Loaned Item

Required Fields:

- NCIPItem item – The item which was loaned to a patron
- Date dateDue – The item’s due date

NCIP User Requested Item

Required Fields:

- NCIPItem item – The item which was requested by a patron
- Date datePlaced – The date when the request was placed
- Date pickupDate – The date when the item is expected to become available for pickup

Optional Fields:

- String status – The status of the request

NCIP Visible User ID

Optional Fields:

- String userId – The user’s ID. This needn’t be a unique ID
- String userIdType – The type of ID being used
- NCIPAgency agency – The agency with which the ID identifies the user

XC User Recalled Item

Required Fields:

- NCIPItem item – The item which was recalled by a patron
- Date datePlaced – The date when the recall was placed
- Date pickupDate – The date when the item is expected to become available for pickup

XC Item Availability

Required Fields:

- String itemId – The ID of the item to which the availability is relevant
- Availability availability – A member of the Availability enumeration defined in Constants.java representing the availability of the item.

Error Reporting

An ILSInterface class is able to report errors by throwing ILSExceptions and AuthenticationExceptions. An AuthenticationException should be thrown whenever the class needs to perform an operation which requires the user to have authenticated and the user has not yet been authenticated. AuthenticationExceptions must be created using their constructor which takes a single String as its only parameter, and this String must be a user friendly error message explaining the cause of the error.

An ILSException should be thrown to report any other error which prevented the ILSInterface class from completing the requested functionality. ILSExceptions should be created using their constructor which takes two Strings and a StrictNcipError Object as its parameters. The first String must be a user friendly error message explaining the cause of the error, and the second String must be an error code describing the error. The next section explains the error codes, and the section after that explains the StrictNcipError class.

Error Codes

Error codes can be accessed from static final Strings in the Constants.java file. Error code constants are stored in such a variable with the name “ERROR_<errorName>”. For example, the error code for the error “0102 - Unsupported Service” is stored in the variable ERROR_UNSUPPORTED_SERVICE. Each error code begins with a 4 digit error number uniquely identifying the error, followed by a brief description of the error. The first two digits in the error number designate the error’s category, and the last two digits identify the specific error within that category. The following error codes are available for use within the NCIP Toolkit:

Internal errors (errors not caused by incoming requests or errors in the NCIP Toolkit configuration file)

- “0001 – Internal Error” A serious error most likely resulting from a bug within the ILSInterface class itself

NCIP Toolkit Developer Documentation

- “0002 – ILS Interface Error” An error caused because the specific ILS instance which the ILSInterface object was interacting with did not behave as expected.

Generic NCIP Request errors

- *”0101 – Request Not Found” An error which occurred because the Request Listener could not find the NCIP Request in the POST data.
- *”0102 – Unsupported Service” An error caused because the incoming NCIP request was of an unrecognized type.
- *”0103 – XML Parse Error” An error signifying that the passed NCIP request was not valid XML.
- *”0104 – NCIP Parse Error” An error signifying that the passed NCIP request was not valid according to the NCIP standard.
- ”0105 – User Not Found” An error caused because an NCIPUser object passed to an ILSInterface method methods was not known to the ILS the ILSInterface object is connecting with.
- ”0106 – Item Not Found” An error caused because an NCIPItem object passed to an ILSInterface method methods was not known to the ILS the ILSInterface object is connecting with.
- ”0107 – Invalid Location” An error caused because a location passed to one of the ILSInterface methods was not known to the ILS the ILSInterface object is connecting with.

Configuration file errors

- ”0201 – NCIP Toolkit Configuration Error” An error caused because one or more values in the configuration file were incorrect.

Authentication errors

- ”0301 – Failed Login” An error generated when the user tries to authenticate with invalid parameters.
- ”0302 – Login Required” An error generated when the ILSInterface object needs to execute functionality which requires the user to be authenticated and the user has not been authenticated. This error code is automatically associated with AuthenticationExceptions.

Request Item Service errors

- ”0401 – Item Not Requestable” An error caused because the requestItem method was called for an item which was an invalid target for requests
- ”0402 – Request Failed” An error signifying that the ILSInterface object tried to place a request on an item and failed.

Cancel Request Item Service errors

- ”0501 – User Has Not Requested Item” An error generated when the cancelRequest method gets called for an item which the user has not placed a request for.
- ”0502 – Cancel Request Failed” An error signifying that the ILSInterface object tried to cancel a request on an item and failed.

Recall Item Service errors

- “0601 – Item Not Recallable” An error caused because the recallItem method was called for an item which was an invalid target for recalls
- “0602 – Recall Failed” An error signifying that the ILSInterface object tried to place a recall on an item and failed.

Cancel Recall Item Service errors

- “0701 – User Has Not Recalled Item” An error generated when the cancelRecall method gets called for an item which the user has not placed a recall for.
- “0702 – Cancel Recall Failed” An error signifying that the ILSInterface object tried to cancel a recall on an item and failed.

Renew Item Service errors

- “0801 – Item Not Checked Out” An error generated when the renewItem method gets called for an item which is not checked out to the user.
- “0802 – Renew Failed” An error signifying that the ILSInterface object tried to renew an item and failed.

* Indicates an error code which is not expected to be used by an ILSInterface class

The writer of an ILSInterface class may create new error codes, but only when the ones listed above fail to adequately describe an error condition. To do this, the new error code should be added to the Constants.java file using the format described above. Wherever possible the first two digits in the error code should be made to match an existing error category.

The StrictNcipError Object

StrictNcipError is a class used by the NCIP Toolkit to display an NCIP Problem element as defined in the NCIP standard. It’s constructor requires a boolean which is true if the error is a Processing Error and false if it’s a Messaging Error, a String containing the error’s scheme, a String containing the error’s type, a String containing the path to the error, and a String containing the value of the element which caused the error or null if the error wasn’t the result of a specific value.

Since the NCIP Request is parsed before any methods are called on the ILSInterface Object, when the ILSInterface class constructs a StrictNcipError it should always create a Processing Error (Messaging Errors are not needed here since they refer to parse errors.) The scheme will either be “NCIP General Processing Error (denoted by a “(G)” in the table below) or “NCIP <request_type> Processing Error” where <request_type> is the type of request that generated the error. For example, the NCIP Lookup User Processing Error scheme contains all the errors which could appear in a Lookup User request.

The following table maps the NCIP Toolkit’s error codes to the StrictNcipError type, path to error, and value which should be used. The path to error refers to a path through the elements in the NCIP request which does not necessarily start from the root element and ends at the element whose

NCIP Toolkit Developer Documentation

contents caused the error. It is typically best to use the path to error exactly as provided in the table below. If there are multiple mappings for a single error code, the StrictNcipError will differ depending on which NCIP request was placed on what information was provided.

Error Code	Type	Path to error	Value
0001	Temporary Processing Failure (G)	NCIPMessage	null
0002	Temporary Processing Failure (G)	NCIPMessage	null
0101	Not used by ILSInterface classes		
0102	Unsupported Service	NCIPMessage	null
0103	Not used by ILSInterface classes		
0104	Not used by ILSInterface classes		
0105	Unknown User	UniqueUserId/UserIdentifierValue	The user ID
0106	Unknown Item	UniqueItemId/ItemIdentifierValue	The item ID (Always use this if the item ID is known)
		UniqueBibliographicId/BibliographicRecordIdentifier	The bibliographic ID (if not in LookupItem)
		VisibleItemId/VisibleItemIdentifier	The bibliographic ID or the holdings ID (if in LookupItem)
0107	Element Rule Violated	ShippingInformation/PhysicalAddress	The location which was not found
0201	Temporary Processing Failure (G)	NCIPMessage	null
0301	User Authentication Failed	AuthenticationInput/AuthenticationInputData	The username they tried to authenticate
0302	Needed Data Missing (G)	NCIPMessage	null
0401	Element Rule Violated	UniqueItemId/ItemIdentifierValue	The item ID (Always use this if the item ID is known)
		UniqueBibliographicId/BibliographicRecordIdentifier	The bibliographic ID
0402	Item Access Denied	UniqueItemId/ItemIdentifierValue	The item ID (Always use this if the item ID is known)
		UniqueBibliographicId/BibliographicRecordIdentifier	The bibliographic ID
0501	Unknown Request	UniqueItemId/ItemIdentifierValue	The item ID
0502	Unknown Request	UniqueItemId/ItemIdentifierValue	The item ID
0601	Item Cannot Be Recalled	UniqueItemId/ItemIdentifierValue	The item ID
0602	Item Cannot Be Recalled	UniqueItemId/ItemIdentifierValue	The item ID
0701	Recall Cannot Be Cancelled At This Time	UniqueItemId/ItemIdentifierValue	The item ID
0702	Recall Cannot Be Cancelled At This Time	UniqueItemId/ItemIdentifierValue	The item ID
0801	Item Not Checked Out	UniqueItemId/ItemIdentifierValue	The item ID
0802	Item Not Renewable	UniqueItemId/ItemIdentifierValue	The item ID

ILSInterface methods

```
public NCIPUser authenticateUser(String username, String password, String agency)
throws ILSException;
```

This method authenticates a user directly against the ILS. The username and password parameters are whichever credentials required to login to the ILS, for example in Voyager the username is the patron's barcode and the password is their last name. The agency which should be authenticated against is also provided.

If the user could be authenticated with the provided credentials, this method must return an NCIPUser object whose `userId` and `usersAgency` fields uniquely identify the user who was authenticated. If the authentication failed, this method must throw an `ILSEException` whose message is a user friendly explanation of why the authentication failed (for example: "The username or password was incorrect.")

If authentication succeeded when this method was called, any subsequent method calls to the same `ILSInterface` Object must behave as if the user was logged into the ILS. The `ILSInterface` class should keep track of the `userId` of the user who was authenticated so it can continue to deny access to other users. It may assume that only one user will be authenticated at a time on each `ILSInterface` Object, and may unauthenticate an authenticated user should a different user authenticate successfully.

```
public NCIPUser authenticateUserAgainstExternalLDAP(String username, String password, String agency) throws ILSEException;
```

This method authenticates a user on behalf of the ILS using an external LDAP server to confirm authentication. It should take advantage of the following parameters in the configuration file to assist with authentication:

- `ExternalLDAPLocation` – The URL at which the LDAP server can be accessed
- `ExternalLDAPPort` – The port at which the LDAP server can be accessed
- `ExternalLDAPStart` – The domain of the LDAP server which should be queried (ex: "ou=patrons, dc=library, dc=edu")
- `ExternalLDAPUsernameAttribute` – The attribute which contains the field the username is expected to belong to
- `ExternalLDAPUserId` – The field on the LDAP server containing the user ID which uniquely identifies the user. The value of this field should be the `userId` on the `NCIPUser` returned by this method

This method may throw an `ILSEException` with a user friendly message explaining that external authentication is not supported if the target ILS does not support external authentication. However, if the target ILS does support external authentication and the external LDAP server parameters in the configuration file were set, this method must attempt to authenticate against the LDAP server before throwing any exceptions.

If the user could be authenticated with the provided credentials, this method must return an `NCIPUser` object whose `userId` and `usersAgency` fields uniquely identify the user who was authenticated. If the authentication failed, this method must throw an `ILSEException` whose message is a user friendly explanation of why the authentication failed (for example: "The username or password was incorrect.")

NCIP Toolkit Developer Documentation

If authentication succeeded when this method was called, any subsequent method calls to the same ILSInterface Object must behave as if the user was logged into the ILS. The ILSInterface class should keep track of the userId of the user who was authenticated so it can continue to deny access to other users. It may assume that only one user will be authenticated at a time on each ILSInterface Object, and may unauthenticate an authenticated user should a different user authenticate successfully.

```
public NCIPUser lookupUser(NCIPUser user, boolean getBlockOrTrap, boolean
getAddress, boolean getVisibleUserID, boolean getFines, boolean getHolds,
boolean getLoans) throws ILSEException;
```

This method is used to look up details on a user. The NCIPUser passed as a parameter has its usersAgency and userId fields set, but all other fields will not be set. This method must return an NCIPUser with the same usersAgency and userId, but with the optional fields set as requested by the caller in the following manner if their values are known:

- If getBlockOrTrap is true, the returned user's list of blocks must contain all blocks currently placed on the user represented by the userId field.
- If getAddress is true, the returned user's address, tempAddress and emailAddress fields must be set to the correct values for the user represented by the userId field if these values are known.
- If getVisibleUserID is true, the user's fullName must be set to the correct value for the user represented by the userId field if this value is known.
- If getFines is true, the returned user's totalFinesCurrencyCode and totalFines fields must be set to the correct values for the user represented by the userId field if these values are known. In addition, the list of fines must contain NCIPUserFine Objects for all fines the user owes.
- If getHolds is true, the returned user's list of holds must contain NCIPUserRequestedItem Objects for all holds and callslips the user represented by the userId field has placed.
- If getLoans is true, the returned user's list of checkedOutItems must contain NCIPUserLoanedItem Objects for all items the user represented by the userId field has checked out.

This method must throw an ILSEException with a user friendly message if anything went wrong which prevented it from accessing information on the user. If the method is able to access user information, but some or all of the requested information is unknown it must return an NCIPUser Object with as many of the requested fields set as it is able to.

```
public NCIPUser xcLookupUser(NCIPUser user, boolean getBlockOrTrap, boolean
getAddress, boolean getVisibleUserID, boolean getFines, boolean getHolds,
boolean getLoans, boolean getRecalls, boolean getMessages) throws
ILSEException;
```

This method functions similarly to lookupUser, except that it allows additional information to be queried which isn't exposed by the NCIP protocol. The NCIPUser passed as a parameter has its usersAgency and userId fields set, but all other fields will not be set. This method must return an NCIPUser with the same usersAgency and userId, but with the optional fields set as requested by the caller in the following manner if their values are known:

NCIP Toolkit Developer Documentation

- If `getBlockOrTrap` is true, the returned user's list of blocks must contain all blocks currently placed on the user represented by the `userId` field.
- If `getAddress` is true, the returned user's `address`, `tempAddress` and `emailAddress` fields must be set to the correct values for the user represented by the `userId` field if these values are known.
- If `getVisibleUserID` is true, the user's `fullName` must be set to the correct value for the user represented by the `userId` field if this value is known.
- If `getFines` is true, the returned user's `totalFinesCurrencyCode` and `totalFines` fields must be set to the correct values for the user represented by the `userId` field if these values are known. In addition, the list of fines must contain `NCIPUserFine` Objects for all fines the user owes.
- If `getHolds` is true, the returned user's list of holds must contain `NCIPUserRequestedItem` Objects for all holds and callslips the user represented by the `userId` field has placed.
- If `getLoans` is true, the returned user's list of `checkedOutItems` must contain `NCIPUserLoanedItem` Objects for all items the user represented by the `userId` field has checked out.
- If `getRecalls` is true, the returned user's list of recalls must contain `XCUserRecalledItem` Objects for all recalls the user represented by the `userId` field has placed.
- If `getMessages` is true, the returned user's list of messages must contain all messages applying the user represented by the `userId` field.

This method must throw an `ILSException` with a user friendly message if anything went wrong which prevented it from accessing information on the user. If the method is able to access user information, but some or all of the requested information is unknown it must return an `NCIPUser` Object with as many of the requested fields set as it is able to.

```
public NCIPItem lookupItem(NCIPItem item, boolean getBibDescription, boolean
getCircStatus, boolean getElectronicResource, boolean getHoldQueueLength,
boolean getItemDescription, boolean getLocation, boolean getCurrentBorrowers,
boolean getCurrentRequesters) throws ILSException;
```

This method is used to look up details on an item. The `NCIPItem` passed as a parameter has its `holdingAgency` and either its `bibId`, `holdingsId` or `itemId` set correctly, but all other fields will not be set. This method must return an `NCIPItem` with the same `holdingAgency` and ID, but with the optional fields set as requested by the caller in the following manner if their values are known:

- If `getBibDescription` is true, the returned item's `author`, `ISBN`, `medium type`, `publisher`, `series` and `title` are set to the correct values for the item represented by the `bibId` field if these values are known.
- If `getCircStatus` is true, the item's `circulation status` is set to the correct values for the item represented by the `bibId` field if it is known.
- If `getElectronicResource` is true, the item's `electronic resource` is set to the correct values for the item represented by the `bibId` field if it is known.
- If `getHoldQueueLength` is true, the item's `hold queue length` is set to the correct values for the item represented by the `bibId` field if it is known.

NCIP Toolkit Developer Documentation

- If getItemDescription is true, the item's title and call number are set to the correct values for the item represented by the bibld field if these values are known.
- If getLocation is true, the item's location is set to the correct values for the item represented by the bibld field if it is known.
- If getCurrentBorrowers is true, the item's list of current borrowers must contain all users who currently have a copy of the item checked out.
- If getCurrentRequesters is true, the item's list of current requesters must contain all users who have placed unfilled requests for the item.

This method must throw an ILSEException with a user friendly message if anything went wrong which prevented it from accessing information on the item. If the method is able to access item information, but some or all of the requested information is unknown it must return an NCIPItem Object with as many of the requested fields set as it is able to.

```
public ArrayList<XCItemAvailability> xcGetAvailability(ArrayList<String>
itemIds) throws ILSEException;
```

This method is used to check the availability of a list of items using the Unique ID element or a list of bibliographic or holdings records using the Visible ID element in the NCIP request sent. The list of Strings passed to this method contains the IDs (could be item ID, Bib or holdings IDs) of each item whose availability is to be returned. The program determines the passed ID to this method and accordingly finds if any corresponding Holdings, Bib or Item ID exist for it. This method must return one XCItemAvailability Object for each item ID in the list passed as a parameter. (There is a feature request to find the XCItemAvailability for each of the item ID's associated of it. It would be incorporated in the 0.5 version).

An XCItemAvailability in the returned list should have its itemId set to the item ID of the item it applies to. If the item does not exist in the Database, the XCItemAvailability's availability would be set to "Does not exist". If the item is available to all patrons, the XCItemAvailability's availability should be set to "Available." If the item is unavailable to all patrons, its availability should be set to "Not Available." If the item is available to some patrons and unavailable to others, its availability should be set to "Possibly Available." Its availability may be set to "Unknown" if there is not enough information to determine the item's availability.

Also, various other information relating to that item (like NCIP Item) is displayed in the response to the request.

This method must throw an ILSEException with a user friendly message if anything went wrong which prevented it from getting the availability on the requested items.

```
public Date renewItem(NCIPUser requestingUser, NCIPItem requestedItem, Date
desiredDueDate) throws AuthenticationException, ILSEException;
```

NCIP Toolkit Developer Documentation

This method is used to renew an item on behalf of a user. The NCIPUser and NCIPItem passed to this method will only their required fields set. The passed Date may be null. If it is not null, it contains the day the user would prefer as the item's new due date.

If the item is able to be renewed, this method must take the required actions to renew it on the ILS. In this case, the item's new due date must be returned.

If this method was called and the requestingUser has not been authenticated, this method must throw an AuthenticationException with a user friendly message explaining that the user must login before they may renew items. If the item cannot be renewed, this method must throw an ILSEException with a user friendly error message explaining why the item could not be renewed unless the underlying ILS allows items to be renewed without the user authenticating themselves.

```
public Date xcOpenUrlRenewItem(NCIPUser requestingUser, String  
openUrlQueryString, Date desiredDueDate) throws AuthenticationException,  
ILSEException;
```

This method is used to renew an item on behalf of a user. The NCIPUser passed to this method will only their required fields set. The passed Date may be null. If it is not null, it contains the day the user would prefer as the item's new due date.

The difference between this method and renewItem is that this one identifies an item with an open URL Query String. If the ILS running alongside the NCIP Toolkit does not contain an open URL resolver then this method may throw an Unsupported Service ILSEException with a message explaining that the underlying ILS cannot process open URL requests.

If the item is able to be renewed, this method must take the required actions to renew it on the ILS. In this case, the item's new due date must be returned.

If this method was called and the requestingUser has not been authenticated, this method must throw an AuthenticationException with a user friendly message explaining that the user must login before they may renew items. If the item cannot be renewed, this method must throw an ILSEException with a user friendly error message explaining why the item could not be renewed unless the underlying ILS allows items to be renewed without the user authenticating themselves.

```
public Date requestItem(NCIPUser requestingUser, NCIPItem requestedItem,  
String comments, String pickupLocation, Date pickupExpiry, boolean  
isCallslip) throws AuthenticationException, ILSEException;
```

This method is used to place a request for an item on behalf of a user. The NCIPUser and NCIPItem passed to this method will only their required fields set, although the NCIPItem may have its bibld set instead of its itemId. The comments, pickupLocation, and pickupExpiry passed to this method may be null. If they are not null, they represent comments about how the item should be delivered, the location the user wishes to pick up the item, and the date after which the user no longer needs the item respectively.

NCIP Toolkit Developer Documentation

The `isCallslip` boolean specifies the type of request to be placed. If it is true, the ILS should submit a callslip request for the item, which is a request that an item be delivered to a location other than its current location and be held there for pickup. If it is false, the ILS should submit a hold request for the item, which simply holds the item for pickup at its current location. Some ILSs differentiate these two requests (such as Voyager,) and on these ILSs the correct request must be placed if possible.

If this method was called and the passed user has not been authenticated, this method must throw an `AuthenticationException` with a user friendly message explaining that the user must login before they may request items. If the item cannot be requested, this method must throw an `ILSException` with a user friendly error message explaining why the item could not be requested unless the underlying ILS allows items to be requested without the user authenticating themselves.

```
public String xcOpenUrlRequestItem(NCIPUser requestingUser, String
openUrlQueryString, String comments, String pickupLocation, Date
pickupExpiry) throws AuthenticationException, ILSException;
```

This method is used to place a request for an item on behalf of a user. The `NCIPUser` passed to this method will only their required fields set. The comments, pickupLocation, and pickupExpiry passed to this method may be null. If they are not null, they represent comments about how the item should be delivered, the location the user wishes to pick up the item, and the date after which the user no longer needs the item respectively.

The difference between this method and `requestItem` is that this one identifies an item with an open URL Query String. If the ILS running alongside the NCIP Toolkit does not contain an open URL resolver then this method may throw an `UnsupportedService ILSException` with a message explaining that the underlying ILS cannot process open URL requests.

If this method was called and the passed user has not been authenticated, this method must throw an `AuthenticationException` with a user friendly message explaining that the user must login before they may request items. If the item cannot be requested, this method must throw an `ILSException` with a user friendly error message explaining why the item could not be requested unless the underlying ILS allows items to be requested without the user authenticating themselves.

```
public NCIPItem cancelRequest(NCIPItem item, NCIPUser user) throws
AuthenticationException, ILSException;
```

This method is used to cancel a request for an item placed by a user. The `NCIPUser` and `NCIPItem` passed to this method will only their required fields set.

If the request is able to be cancelled, this method must take the required actions to cancel it on the ILS. In this case, the item whose request was cancelled is returned to confirm success.

If this method was called and the requestingUser has not been authenticated, this method must throw an `AuthenticationException` with a user friendly message explaining that the user must login before they may cancel requests. If the request for the item cannot be cancelled, this method must throw an `ILSException` with a user friendly error message explaining why the request could not be

NCIP Toolkit Developer Documentation

cancelled unless the underlying ILS allows requests to be canceled without the user authenticating themselves.

```
public NCIPItem recallItem(NCIPItem item, Date desiredDueDate, String  
comments, String pickupLocation, Date pickupExpiry) throws  
AuthenticationException, ILSEException;
```

This method is used to recall an item on behalf of a user. The NCIPUser and NCIPItem passed to this method will only their required fields set. The comments, pickupLocation, and pickupExpiry passed to this method may be null. If they are not null, they represent comments about how the item should be delivered, the location the user wishes to pick up the item, and the date after which the user no longer needs the item respectively.

If the item is able to be recalled, this method must take the required actions to recall it on the ILS. In this case, the item's expected availability date must be returned. The item may not actually become available on this date for a variety of reasons (such as the item not being returned in time.) Rather, this is a reasonable estimate of when the item will be available to the user.

If this method was called and no user has been authenticated, this method must throw an AuthenticationException with a user friendly message explaining that the user must login before they may renew items. If the item cannot be recalled, this method must throw an ILSEException with a user friendly error message explaining why the item could not be recalled unless the underlying ILS allows items to be recalled by an anonymous user.

```
public NCIPItem cancelRecall(NCIPItem item) throws AuthenticationException,  
ILSEException;
```

This method is used to cancel a recall of an item placed by the currently authenticated user. The NCIPItem passed to this method will only their required fields set.

If the recall is able to be cancelled, this method must take the required actions to cancel it on the ILS. In this case, the item whose recall was cancelled is returned to confirm that the operation was successful.

If this method was called and no user has been authenticated, this method must throw an AuthenticationException with a user friendly message explaining that the user must login before they may cancel recalls. If the recall of the item cannot be cancelled, this method must throw an ILSEException with a user friendly error message explaining why the recall could not be cancelled unless the underlying ILS allows recalls to be canceled by an anonymous user.

```
public NCIPItem createItem(NCIPItemOptionalFields data) throws ILSEException;
```

This method is used to create a new item on the ILS. The passed NCIPItemOptionalFields contains all the information the client has about the item to be created. If the item on the NCIPItemOptionalFields is set, its item ID is the item ID which the client is proposing the NCIP Toolkit

NCIP Toolkit Developer Documentation

assign to the item; the NCIP Toolkit may use a different item ID if appropriate (for example, if the proposed ID was assigned to a different item.)

If the item was successfully created, this method should return an NCIPItem with the item ID it assigned to the newly created item. Otherwise it should throw an ILSEException explaining why the item could not be created.

```
public NCIPCheckOutItemResponse checkOutItem(NCIPUser user, NCIPItem item,  
int acknowledgedFeeAmount, String acknowledgedFeeCurrencyCode, List<String>  
acknowledgedItemUseRestrictionTypes, Date desiredDueDate,  
NCIPRequestedInformation requestedInfo, Date mandatedAction, boolean  
resourceDesired, NCIPUserAddressInformation shippingAddressInfo, String  
shippingInstructions, String shippingNote, NCIPRequestId requestId) throws  
AuthenticationException, ILSEException;
```

This method is used to check out an item on behalf of a user. The passed user and item are the user checking out the item and the item being checked out. If the user is acknowledging any fees they owe or restrictions on the item's use they are shown in the acknowledgedFeeAmount, acknowledgedFeeCurrencyCode and acknowledgedItemUseRestrictionTypes. The desiredDueDate is the date when the user would like the item to be due. The requestedInfo parameter lists the information about the user or the item that should be included in the response. If resourceDesired is true, the response should include a reference to the resource. shippingAddressInfo, shippingInstructions and shippingNote outline where the item should be sent and any additional instructions for shipping the item. If the client who placed the request has already updated their records assuming the item was checked out to the user, mandatedAction is the date when its records were updated.

The NCIPCheckOutItemResponse class (which this method returns an Object of if successful) contains numerous fields which can be used to send information back to the client. The returned object must have its item field set to the item which was checked out. Any number of other fields may also be set, and this method should set fields containing information the caller requested with the requestedInfo parameter. If the item could not be checked out, this method should throw an ILSEException explaining what went wrong.

```
public NCIPCheckInItemResponse checkInItem(NCIPItem item, Date  
mandatedAction, NCIPRequestedInformation requestedInfo) throws ILSEException;
```

This method is used to check in an item indicated by the item parameter. If the client who placed the request has already updated their records assuming the item was checked in, mandatedAction is the date when its records were updated. The requestedInfo parameter lists the information about the item and the user it was checked out to that should be included in the response.

The NCIPCheckInItemResponse class (which this method returns an Object of if successful) contains numerous fields which can be used to send information back to the client. The returned object must have its item field set to the item which was checked in. Any number of other fields may also be set, and this method should set fields containing information the caller requested with the

NCIP Toolkit Developer Documentation

requestedInfo parameter. If the item could not be checked in, this method should throw an `ILSException` explaining what went wrong.

```
public NCIPAcceptItemResponse acceptItem(NCIPRequestId request, String
requestedActionType, Date dateForReturn, boolean indeterminateLoanPeriod,
boolean nonReturnable, Date mandatedAction, NCIPItem item,
NCIPItemOptionalFields itemFields, NCIPUserOptionalFields userFields,
NCIPUser user, NCIPFiscalTransactionInformation fiscalTransaction, boolean
renewalNotPermitted) throws ILSException;
```

This method is called to request that an item be accepted for an action, such as checking it out to a user. The item and the user need not be known to the system when this method is called. At minimum the request and requestedActionType parameters must not be null and must tell what request should be accepted. If the item or the user is not known, the item, itemFields, user and userFields parameters may describe the item and user involved in the request to be accepted. The fiscalTransaction parameter provides information on any financial transaction which took place to enable the request. If the request is a circulation request, the dateForReturn, indeterminateLoanPeriod, nonReturnable, and renewalNotPermitted parameters may provide additional information about the request. If the client who placed the request has already updated their records assuming the item was accepted, mandatedAction is the date when its records were updated.

The NCIPAcceptItemResponse class (which this method returns an Object of if successful) must contain the request ID of the request that was accepted. It may also contain the item ID of the item the request was accepted for. If the item could not be accepted for the request, this method should throw an `ILSException` explaining what went wrong.

Example Implementation: The VoyagerInterface

The VoyagerInterface project is an ILSInterface implementation that was created to connect to the Voyager ILS. For detailed instructions on how to set up the NCIPToolkit please refer to the User Documentation.

The following configuration parameters were added to the driver_config.xml configuration file:

- VoyagerUrl – The URL where Voyager can be accessed
- VoyagerDatabaseUrl – The URL where the Voyager database can be accessed
- VoyagerRequestSuccessMessage – A message or block of HTML which is returned by Voyager only when a request form is successfully submitted. The NCIP Toolkit searches the page returned by Voyager after a request form is submitted for this value to determine whether or not the request was successful.
- VoyagerValidCircStatusesCallslip – A comma separated list containing all circulation statuses an item may have to be the valid target of a callslip request

NCIP Toolkit Developer Documentation

- `VoyagerValidCircStatusesHold` – A comma separated list containing all circulation statuses an item may have to be the valid target of a hold request
- `VoyagerValidCircStatusesRecall` – A comma separated list containing all circulation statuses an item may have to be the valid target of a recall request
- `VoyagerDatabaseName` – The name of the Voyager database
- `VoyagerDatabaseWriteAuthUsername` – The username for a user on the Voyager database which has write access to the `WOPAC_PID_PATRON_KEYS` table
- `VoyagerDatabaseWriteAuthPassword` – The password for a user on the Voyager database which has write access to the `WOPAC_PID_PATRON_KEYS` table
- `VoyagerDatabaseReadOnlyUsername` – The username for a user on the Voyager database which has read access to all tables.
- `VoyagerDatabaseReadOnlyPassword` – The password for a user on the Voyager database which has read access to all tables.

The `VoyagerInterface` class was given a static `log4j` Logger Object which is used to write to the NCIP Toolkit's log file (`VoyagerInterface.java` around line 58 creates this Object. This line can and should be copied into any `ILSInterface` class to enable logging.) An `HttpClient` is used to make HTTP requests of Voyager when it is necessary to screenscape the ILS's OPAC (`VoyagerInterface.java` line 81) There are also variables to store the PID and SEQ, which Voyager uses to track an authenticated session, and the user ID of the user who is currently authenticated on the ILS. Since each connection's session stores its own reference to the `RequestHandler` and `ILSInterface`, authentication information may be stored in private variables for the `VoyagerInterface`; different connections will use different `VoyagerInterface` Objects.

One serious issue that needed to be overcome while writing the `VoyagerInterface` class was that Voyager might timeout an authenticated login sooner than the NCIP Toolkit is configured to timeout. To handle this problem, the inner class `VoyagerHeartbeatThread` was created in the `VoyagerInterface.java` file. When a `VoyagerInterface` Object authenticates a user, it creates a `VoyagerHeartbeatThread` for that user. Each time the `VoyagerInterface` places a request on that user's behalf it signals the `VoyagerHeartbeatThread`, so at any point in time the `VoyagerHeartbeatThread` knows how much time has passed since the last request was placed.

Every minute the `VoyagerHeartbeatThread` checks whether or not the NCIP Toolkit's timeout has expired for the `VoyagerInterface` to which it belongs. If the timeout has not expired, it sends a request to Voyager to load the authenticated user's information page (which is typically easy for Voyager to load.) However, if the NCIP Toolkit's timeout expired it signals the `VoyagerInterface` owning it to delete the authenticated user's authentication information. This prevents Voyager from timing out an authenticated session according to its own timeout, and forces the session to timeout when it has been inactive for the number of seconds defined in the NCIP Toolkit's configuration file.

After that, all that remained was to implement the `ILSInterface` methods in the `VoyagerInterface` class. They were implemented in the following way:

NCIP Toolkit Developer Documentation

```
public NCIPUser authenticateUser(String username, String password)
throws ILSEException;
```

When this method is called, the VoyagerInterface class checks the authentication parameters against the Voyager database to determine whether or not a login is valid. It first checks to see if there are any patrons in the database with an institution_id (user ID) equal to the passed username and a last name equal to the passed password, ignoring case. If there was no match, it then checks whether there are any patrons with a barcode equal to the passed username and a last name equal to the passed password, ignoring case. If either of these checks returned a user, the NCIP Toolkit declares the authentication to be successful, otherwise it throws an ILSEException stating that the username or password was incorrect.

If the authentication was successful, the NCIP Toolkit gets a PID and SEQ for its session with Voyager by parsing the corresponding hidden fields returned by an HTTP request to the main Voyager page. It then authenticates its session with Voyager by writing the institution_id and pid to the WOPAC_PID_PATRON_KEYS table in the Voyager database. The PID, SEQ, and authenticated user's institution ID are stored in private variables to prove that the user was authenticated to future requests of Voyager. This method then creates and returns an NCIPUser Object with the authenticated user's ID.

```
public NCIPUser authenticateUserAgainstExternalLDAP(String username, String
password) throws ILSEException;
```

This method tries to authenticate the user against the LDAP server defined in the configuration file, and throws an ILSEException stating that the username or password was incorrect if the authentication failed. Otherwise, the NCIP Toolkit gets a PID and SEQ for its session with Voyager by parsing the corresponding hidden fields returned by an HTTP request to the main Voyager page. It then authenticates its session with Voyager by writing the institution_id and pid to the WOPAC_PID_PATRON_KEYS table in the Voyager database. The PID, SEQ, and authenticated user's institution ID are stored in private variables to prove that the user was authenticated to future requests of Voyager. This method then creates and returns an NCIPUser Object with the authenticated user's ID.

```
public NCIPUser lookupUser(NCIPUser user, boolean getBlockOrTrap, boolean
getAddress, boolean getVisibleUserID, boolean getFines, boolean getHolds,
boolean getLoans) throws ILSEException;
```

This method queries the Voyager database to get information on the user depending on which of the boolean parameters are set to true. The information is set on the passed NCIPUser Object, which is then returned.

The exception is block or trap information, which generated dynamically by the Voyager OPAC instead of being stored in the database. Because of this, the only way to get block or trap information is to screenscape the user's information page in Voyager. For this reason block or trap information is not returned if the requested user is not authenticated, even if getBlockOrTrap is set to true.

```
public NCIPUser xcLookupUser(NCIPUser user, boolean getBlockOrTrap, boolean
getAddress, boolean getVisibleUserID, boolean getFines, boolean getHolds,
```

NCIP Toolkit Developer Documentation

```
boolean getLoans, boolean getRecalls, boolean getMessages) throws  
ILSEException;
```

This method queries the Voyager database to get information on the user depending on which of the boolean parameters are set to true. The information is set on the passed NCIPUser Object, which is then returned.

The exception is block or trap information, which generated dynamically by the Voyager OPAC instead of being stored in the database. Because of this, the only way to get block or trap information is to screenscape the user's information page in Voyager. For this reason block or trap information is not returned if the requested user is not authenticated, even if getBlockOrTrap is set to true.

```
public NCIPItem lookupItem(NCIPItem item, boolean getBibDescription, boolean  
getCircStatus, boolean getElectronicResource, boolean getHoldQueueLength,  
boolean getItemDescription, boolean getLocation, boolean getCurrentBorrowers,  
boolean getCurrentRequesters) throws ILSEException;
```

This method queries the Voyager database to get information on the item depending on which of the boolean parameters are set to true. The information is set on the passed NCIPItem Object, which is then returned.

```
public ArrayList<XCItemAvailability> xcGetAvailability(ArrayList<String>  
itemIds) throws ILSEException;
```

When this method is called, each of the item IDs(or bib/holdings using the VisibleItemId element) is looked up in the Voyager database's circulation matrix. An XCItemAvailability Object is added to an array of XCItemAvailability Objects for each item ID with its availability set to Does not Exist if it does not exist in the Voyager Database, Available if all patrons can request that item and its circulation status renders it valid for Callslip requests, Unavailable if no patrons can request that item or its circulation status renders it invalid for Callslip requests, Possibly Available if some but not all patron groups can request that item, and Unknown if there were no entries in the circulation matrix for that item. The list of XCItemAvailability Objects is then returned. Also, more information about each of the item/bib/holdings ID is sought as similar to the NCIPLookupItem response is sent back in the NCIP response for this verb.

```
public Date renewItem(NCIPUser requestingUser, NCIPItem requestedItem, Date  
desiredDueDate) throws AuthenticationException, ILSEException;
```

First, this method confirms that the passed NCIPUser has been authenticated on the VoyagerInterface Object, and throws an ILSEException saying that the user must log in if they haven't.

Voyager's user information page has a form with a list of checked out items with a checkbox next to each and a button to renew checked items. When the VoyagerInterface's renewItem method is called, it first looks up the title of the item based on the passed NCIPItem's item ID. If the item was not found in the database it throws an ILSEException indicating that the item didn't exist. Otherwise it compares the item's title with the list of items the user has checked out. If the title was not found in this list, an ILS Exception is thrown saying that the user does not have the item checked out. Otherwise it screenscrapes the Voyager information page, submitting the form to renew the item with the correct

item selected. Finally, it parses the user's information page to get the item's new due date and returns it.

Since Voyager does not have an option to submit a desired due date when renewing an item, the Date parameter passed to this method is ignored. However, other ILSInterface implementations should make use of it if the corresponding ILS supports this functionality.

```
public Date xcOpenUrlRenewItem(NCIPUser requestingUser, String  
openUrlQueryString, Date desiredDueDate) throws AuthenticationException,  
ILSException;
```

Voyager does not contain an open URL resolver, so this method throws an ILS Exception explaining that the service is not supported.

```
public Date requestItem(NCIPUser requestingUser, NCIPItem requestedItem,  
String comments, String pickupLocation, Date pickupExpiry, boolean  
isCallslip) throws AuthenticationException, ILSException;
```

First, this method confirms that the passed NCIPUser has been authenticated on the VoyagerInterface Object, and throws an ILSException saying that the user must log in if they haven't.

Next, this method checks the item's circulation status in the Voyager database against a list of valid circulation statuses for the type of request (callslip or hold) defined in the configuration file. It also checks the circulation policy matrix to determine whether or not the authenticated user is able to request the item. If the circulation status is not valid for the request or the authenticated user is not permitted to place the request, an ILSException is thrown explaining this.

If the user is able to request the item, the VoyagerInterface makes an HTTP request to load the Voyager request form for callslips or holds depending on the value of the isCallslip boolean. It fills out the appropriate fields on the form and submits it. The NCIP Toolkit configuration file contains a string which appears in a successful response to a request form and does not appear in a failed response. If this string appears, the date the requested item will become available is returned to indicate success, otherwise an ILSException is thrown which says that the request failed.

```
public String xcOpenUrlRequestItem(NCIPUser requestingUser, String  
openUrlQueryString, String comments, String pickupLocation, Date  
pickupExpiry) throws AuthenticationException, ILSException;
```

Voyager does not contain an open URL resolver, so this method throws an ILS Exception explaining that the service is not supported.

```
public NCIPItem cancelRequest(NCIPItem item, NCIPUser user) throws  
AuthenticationException, ILSException;
```

First, this method confirms that the passed NCIPUser has been authenticated on the VoyagerInterface Object, and throws an ILSException saying that the user must log in if they haven't.

Voyager's user information page has a form with a list of requested items with a checkbox next to each and a button to cancel checked requests. When the VoyagerInterface's cancelRequest method is called, it first looks up the title of the item based on the passed NCIPItem's item ID. If the item was

NCIP Toolkit Developer Documentation

not found in the database it throws an `ILSEException` indicating that the item didn't exist. Otherwise it compares the item's title with the list of items the user has requested. If the title was not found in this list, an `ILS Exception` is thrown saying that the user has not requested the item. Otherwise it screenscrapes the Voyager information page, submitting the form to cancel the request with the correct item selected. Finally, this method returns the `NCIPItem` Object that was passed to it.

```
public NCIPItem recallItem(NCIPItem item, Date desiredDueDate, String  
comments, String pickupLocation, Date pickupExpiry) throws  
AuthenticationException, ILSEException;
```

First, this method confirms that the passed `NCIPUser` has been authenticated on the `VoyagerInterface` Object, and throws an `ILSEException` saying that the user must log in if they haven't.

Next, this method checks the item's circulation status in the Voyager database against a list of valid circulation statuses for recalls defined in the configuration file. It also checks the circulation policy matrix to determine whether or not the authenticated user is able to recall the item. If the circulation status is not valid for recalls or the authenticated user is not permitted to place the recall, an `ILSEException` is thrown explaining this.

If the user is able to recall the item, the `VoyagerInterface` makes an HTTP request to load the Voyager request form for recalls. It fills out the appropriate fields on the form and submits it. The NCIP Toolkit configuration file contains a string which appears in a successful response to a request form and does not appear in a failed response. If this string appears, the date the recalled item will be available returned to indicate success, otherwise an `ILSEException` is thrown which says that the recall failed.

```
public NCIPItem cancelRecall(NCIPItem item) throws AuthenticationException,  
ILSEException;
```

First, this method confirms that the passed `NCIPUser` has been authenticated on the `VoyagerInterface` Object, and throws an `ILSEException` saying that the user must log in if they haven't.

Voyager's user information page has a form with a list of recalled items with a checkbox next to each and a button to cancel checked recalls. When the `VoyagerInterface`'s `cancelRecall` method is called, it first looks up the title of the item based on the passed `NCIPItem`'s item ID. If the item was not found in the database it throws an `ILSEException` indicating that the item didn't exist. Otherwise it compares the item's title with the list of items the user has recalled. If the title was not found in this list, an `ILS Exception` is thrown saying that the user has not requested the item. Otherwise it screenscrapes the Voyager information page, submitting the form to cancel the recall with the correct item selected. Finally, this method returns the `NCIPItem` Object that was passed to it.