KINETIC LINK

User's Guide
Version 1.0.0

Revised August 28, 2006

Project Supported By:

Kinetic Data, Inc.
235 East 6th Street
St. Paul, MN 55101
651-695-8566
www.kineticdata.com

Contents

Chapter 1 Klink Overview

What is Klink?

Kinetic Link, or Klink, is a web framework that provides an HTTP interface for interacting with backend data sources. The framework itself is written using Java Servlets and Apache Struts, however once Klink is running any language that can make HTTP requests and parse the xml results can be used to consume the framework.

What is Klink-Ars?

The design of Klink allows it to be extended to interact with just about any backend data source. Currently, the only data source supported is the Action Request System (commonly referred to as Remedy). Klink-Ars is the version of Klink that provides the ability to interact with Ars Servers. In the future, additional versions of Klink may be released providing the ability to interact with SAP, HP Service Desk, or directly with databases. For the purpose of this document, the term Klink will be used to refer to the Ars distribution.

How does Klink work?

At its heart, Klink is an application that sits on top of a web server capable of serving Java Servlets. Requests are made by submitting an HTTP request and specifying required information as part of the URL, HTTP header information, or as HTTP parameters as required by that particular framework call. These requests function very similar to the standard requests made by browsers to retrieve web pages, in fact any web browser can be used to begin interacting with a backend data source through Klink.

Why use Klink?

Klink was initially developed to provide an easy, extensible, and flexible way to interact with Remedy. Remedy provides two basic methodologies (a Java Wrapper API and Web Services) for external interaction when the User Tool and Mid-tier are not sufficient. Unfortunately, the API is difficult to work with and manage and the Remedy Web Service support is extremely limited. Kinetic Link is a way of interacting with Remedy systems without the hassle of dealing directly with the API or working around the limitations of web services.

Support

For support please visit the support forums located on the Klink homepage (*http://www.kineticfoundation.org/klink*), or contact the main development team at klink@kineticdata.com.

Chapter 2 Klink Design

Calling Klink

The Klink framework is called by making an HTTP request to a URL similar to *http://host.domain.com/klink/framework_call*, where host.domain.com is the domain name or IP address of the server hosting the Klink webapp and framework_call is a valid Klink call (for a full list of framework calls, please see Chapter 3 – Klink Framework Calls). The easiest way to test to see if Klink is functioning properly is to make the *about* framework call. For example, pointing a web browser to *http://mybox:8080/klink/about* will call the *about* framework method on the instance of Klink running on the server *mybox* port 8080.

Making Calls to Klink-Ars

In order to successfully communicate with a Remedy server Klink needs to know two things: the location of the Remedy Server and a set of valid user credentials. The server information is included as part of the HTTP request URL and the user information can be included either as part of that server information within the URL or as an HTTP header element via HTTP basic authentication. The Klink framework itself is stateless, so server and user information must be sent as part of every framework call.

Specifying the Remedy Server

In general, the Remedy Server information is provided on the request URL after the framework method name in the following format: Server:TcpPort:RpcPort where Server is the domain name or IP address of the Remedy server, TcpPort is the TCP port that the server is listening on, and RpcPort is the RPC port that the server is listening on. For example, the framework call *http://mybox:8080/klink/usercheck/RemServ1:3000:9000* will validate the current user on the Remedy server instance running on RemServ1, TCP port 3000, RPC port 9000. When not provided, the TCP and RPC port default to 0, which attempts to contact the Remedy PortMapper service on the Remedy server to automatically obtain the port information. The call *http://mybox:8080/klink/usercheck/RemServ1* is identical to *http://mybox:8080/klink/usercheck/RemServ1:0* as well as the call *http://mybox:8080/klink/usercheck/RemServ1:0:0*. Therefore, if a TCP port is required to connect to a Remedy server but not an RPC port the server information can be in the following format: Server:TcpPort. If an RPC port must be specified without a TCP port the following syntax can be used: Server:0:RpcPort.

Specifying the Remedy User Credentials

The easiest way to specify user credentials is as part of the server information on the URL. This can be done by prepending user:pass@ to the server information, where user is the username and pass is the password that Klink will use to interact with the Remedy server. For example, *http://mybox:8080/klink/usercheck/Demo:dpass1@RemServ1* will attempt to validate the user *Demo* using *dpass1* as the password on the remedy server RemServ1.

Klink also supports specification of user credentials through HTTP Basic Authentication. When interacting with Klink via a web browser, assuming no special actions are taken, the first time you try to access a non-management framework call (any framework call not used to interact with Klink itself) you will be prompted for a username and password. These credentials are then used as your default user credentials for the length of that browser session. Most browsers allow Authentication to be reset by prepending user:pass@ to the web server. For example, *http://myUser:myPass@mybox:8080/klink/user* will reset your default user credentials to myUser and myPass.

Whenever a non-management framework call is accessed without specifying user credentials (either as part of the server information or via HTTP Basic Authentication) a HTTP 401 Unauthorized response will be returned. When credentials are sent both via HTTP Basic Authentication and as part of the server information the credentials specified as part of the server information take precedence over those specified using basic auth.

Klink Results

All Klink framework calls return an HTTP response with a *text/xml* content type and xml the body of the response. Following is an example of the xml skeleton returned from a framework response:

```
<Response RequestMethod="entries" Success="true">
   <Messages>
      <Message Type="InvalidParameter">
         Unrecognized HTTP parameter "itemss", parameter ignored.
      </Message>
   </Messages>
   <Result>
      ...
   </Result>
</Response>
```

The root element, Response, contains attributes representing the invoking method name and the success of the call as well as the child nodes Messages and Result. The Success attribute will always have the value *true* when the framework call was successful and *false* when there was an error processing the request.

Whenever there is a problem processing the request, a Message will be included as a child element to the Messages node. A Message element includes the MessageType and the actual message. See Generic Klink Messages and Klink-Ars Messages for a description of the types of messages returned in this node.

The Result element will either be null or have a single child element that represents the result of a framework call. See Chapter Three or Appendix B for a complete list of result children.

Generic Klink Messages

Klink can provide the users with three standard message types: InvalidParameter, InternalException, and UnexpectedException. Below is an example of each as well as a short description.

InvalidParameter

An InvalidParameter message is used to alert the Klink user to any invalid framework call parameters that don't prevent the call from completing. Invalid framework call parameters could be unrecognized parameter names or invalid parameter values that have fell over to a default.

```
<Message Type="InvalidParameter">
   Unrecognized HTTP parameter "itemss", parameter ignored.
</Message>
<Message Type="InvalidParameter">
   Unable to process the Includenullitems prameter, "truee" is not a valid value.
</Message>
```

InternalException

An InternalException message is used when there was an expected problem handling the framework request. These include problems such as invalid parameter values for parameters without a default, problems connecting to the Remedy server, and problems generating the result element.

```
<Message Type="InternalException">
   Unable to convert the sort items to numerical values, "w" is not a valid number.
</Message>
```

ModelException

A ModelException message is used when there is a problem creating, modifying, or converting any of the result elements or when the entry xml specified for an update or create call is invalid.

```
<Message Type="ModelException">
   Unable to process an entry with conflicting inherent (specified as Entry Xml
   attribute) and explicit structure IDs.
</Message>
```

UnexpectedException

An UnexpectedException message is used when there was an unrecoverable problem handling the framework request. These can occur for values that the Klink framework doesn't validate as well as extreme cases which havn't been encountered yet.

```
<Message Type="UnexpectedException">
   java.lang.IllegalArgumentException: length longer than
   AR_MAX_ENTRYID_SIZE
</Message>
```

Klink-Ars Messages

Klink can also return Ars Messages back to the user. These can be error messages, warning messages, or any messages triggered by a Remedy filter message action. All Message elements created from Ars messages have a type of "ArsMessage" and may have additional attributes such as the message number and Ars message type.

```
<Message MessageNumber="59" Type="NOTE">
   This message was generated by a filter message action!
</Message>
<Message MessageNumber="59" Type="WARNING">
   No such user exists -- successfully connected as a guest user: </Message>
<Message MessageNumber="353" Type="ERROR">
   You have no access to form: User</Message>
</Messages>
```

Chapter 3 Klink Framework Calls

Klink Management Calls

About

Call: /klink/about

This call returns information about the running system. This includes the version information for both Klink and the libraries required for klink to run. Libraries shown as missing will prevent Klink from functioning properly.

Logconfig

Call: /klink/logconfig
Parameters: configfile, configproperties

This framework call is disabled by default. This framework call can be enabled by uncommenting it from the struts-config.xml file. For more information please see the Installation and Configuration Guide. When logconfig is called without specifying a parameter the current log4j configuration parameters are displayed, along with the current source of those configurations. Specifying a configfile will do a live reload (flushing current properties and loading the new) of the logging configuration, loading parameters from the file specified. Properties can be added/modified by hand via the configproperties parameter. Using the configproperties parameter will not force a flush of the current configuration.

Sample1: *http://<klinkhost>/klink/logconfig*
This will display the current log4j configuration parameters.

Sample2: *http://<klinkhost>/klink/logconfig?configfile=DEFAULT*
This will reset the log4j configuration parameters based on the file specified in the application's web.xml file (See the Installation and Configuration Guide for more information).

Sample3: *http://<klinkhost>/klink/logconfig?configfile=C:\log4j\GenericDEBUG.cfg*
This will flush the current log4j configuration parameters and load a new configuration based on the file specified (In this case GenericDEBUG.cfg).

Sample4: *http://<klinkhost>/klink/logconfig?configproperties=
log4j.logger.com.kd.klink:DEBUG*
This will either add or modify the current log4j.logger.com.kd.klink configuration parameter without flushing previous configurations.

Sample5: *http://<klinkhost>/klink/logconfig?configfile=C:\log4j\GenericDEBUGLogging.cfg
&configproperties=log4j.logger.com.kd.klink:DEBUG, fileAppender*
This will flush the current configurations, load the configurations from the configfile specified, and add or modify the current log4j.logger.com.kd.klink configuration parameter.

Meta Information Related Calls

Configurations

Call: /klink/configurations/<Datasource>
Parameters: items

Retrieves configuration information for the Remedy server specified by Datasource. Configuration items are specified by name, a complete list is available by making a configurations call without including the items parameter.

Sample1: *http://<klinkhost>/klink/configurations/<datasource>*
This will return a list of all of the configurations for the <remedyserver> Remedy Server.

Sample2: *http://<klinkhost>/klink/configurations/<datasource>?items=SERVER_ID, SERVER_NAME,SERVER_VERSION*
This will return only the SERVER_ID, SERVER_NAME, and SERVER_VERSION configurations.

Permissions

Call: / klink/permissions/<Datasource>/<Structure>[/<ItemID>[,<ItemID>[...]]]
Parameters: items

Retrieves form and field permissions. The base framework call will return permissions to the form itself. When the items parameter is included the framework call will return form permissions and the permissions for all of the specified ItemIDs. When ItemIDs are specified as a part of the framework call only those specified will be returned, without the form permission information.

Sample1: *http://<klinkhost>/klink/permissions/<datasource>/<structure>*
This will return the permissions for the structure specified.

Sample2: *http://<klinkhost>/klink/permissions/<datasource>/<structure>?items=1,2,3*
This will return the permissions for the structure speciried and the three fields shown.

Sample3: *http://<klinkhost>/klink/permissions/<datasource>/<structure>?items=all*
This will return the permissions for the structure specified and the permissions for all of the structure items associated with that structure.

Sample4: *http://<klinkhost>/klink/permissions/<datasource>/<structure>/1*
This will return the permissions for structure item 1 on the structure specified.

Sample5: *http://<klinkhost>/klink/permissions/<datasource>/<structure>/1,2,3*
This will return the permissions for structure items 1, 2, and 2 on the structure specified.

Statistics

Call: /klink/statistics/<Datasource>

Retrieves statistics information for the Remedy server specified by Datasource. Statistics items are specified by name, a complete list is available by making a statistics call without including the items parameter.

Sample1: *http://<klinkhost>/klink/statistics/<datasource>*
This will return a list of all of the statistics for the <datasource> Remedy Server.

Sample2: *http://<klinkhost>/klink/statistics/<datasource>?items=CURRENT_USERS*
This will return only the statistics specified in the items parameter.

Usercheck

Call: /klink/usercheck/<Datasource>

Validates the user by attempting to log in to the Remedy system. If the user was unable to log in (due to bad password, bad account name on a server with guest accounts disabled) or there were any messages that occured during login (such as warning messages if the user has been logged in as a guest) then these will be included in the Klink response. The result element contains the user credentials used for the login. The below examples are all specifying the user credentials as part of the datasource in the framework call, however usercheck functions the same when they are specified as the HTTP Basic Authentication header.

Sample1: *http://<klinkhost>/klink/user/TestUser:correctpass@<datasource>*
This will return the context info and no messages because everything is correct.

Sample2: *http://<klinkhost>/klink/user/TestUser:wrongpass@<datasource>*
This will return an error message because the password is incorrect.

Sample3: *http://<klinkhost>/klink/user/NoUser@<datasource>*
Assuming NoUser is a not an existing user, this will return the context info and a warning message if guest accounts are enabled or an error message.

Structure Related Calls

Structure

Call: /klink/structure/<Datasource>/<Structure>
Parameters: items

Returns a description of the structure specified. By default, this will describe all data-related structure items. To include non-data-related structure items set the items parameter to "all". A subset of structure items can be retrieved by specifying a list of IDs as the value of the items parameter.

Sample1: *http://<klinkhost>/klink/structure/<datasource>/<structure>*
This returns a description of the data-related structure items on the structure specified.

Sample2: *http://<klinkhost>/klink/structure/<datasource>/<structure>?items=all*
This returns a description of all of the structure items on the structure specified.

Sample3: *http://<klinkhost>/klink/structure/<datasource>/<structure>?items=1,3,7*
This returns a description of the structure items with IDs 1, 3, and 7.

Structures

Call: /klink/structures/<Datasource>

Retrieves a list of all of the structures the requesting user has access to on the Remedy Server.

Sample1: *http://<klinkhost>/klink/structures/<datasource>*

Data Related Calls

Attachment

Call: /klink/attachment/<Datasource>/<Structure>/<EntryID>/<AttachmentItemID>

This returns an attachment entry item and is the only way to retrieve the actually attachment data. Calls to entry will only return the attachment name and size. The data is represented as the file contents base64 encoded. To retrieve the byte stream of the attachment simply decode the value of the data element.

Create

Call: /klink/create/<Datasource>[/<Structure>]
Parameters: entry, returnentry

This will create an entry within the specified structure. The simplest way to specify the entry information is on the Url via the http parameter "entry". However, the best practice for production systems is to set the content type of the HTTP request to be application/xml and include the entry xml as the request body. The format of the entry xml is identical to the xml returned as a result of the "entry" framework call, however only the entry items required for creation need to be present. By default, this call will only return an empty Entry element with the ID and Structure attributes set. If a full view of the EntryItems is required, the "returnentry" parameter can be set to true. The Structure can either be set as an attribute to the Entry element or as part of the request Url. If it appears in both and is different, Klink will return a ModelException.

Sample1: *http://<klinkhost>/klink/create/<datasource>?
entry=<Entry Structure="TestForm"/>*
This will return likely return an error because field 2 (Submitter) and 8 (Short Description) are required and don't have defaults.

Sample2: *http://<klinkhost>/klink/create/<datasource>/TestForm?entry=<Entry Structure="TestForm"><EntryItem ID="2">Bob</EntryItem><EntryItem ID="7">Assigned</EntryItem><EntryItem ID="8">
Description</EntryItem></Entry>*
This will create an entry in the Assigned state (field id 7 is Status).

Sample3: *http://<klinkhost>/klink/create/<datasource>/TestForm?returnentry=true&
entry=<Entry><EntryItem ID="2">$USER$</EntryItem><EntryItem ID="7">Assigned</EntryItem><EntryItem ID="8">
Description</EntryItem></Entry>*
This will create an entry in the Assigned state and return the full description of the entry.

Sample4: *http://<klinkhost>/klink/create/<datasource>/TestForm?returnentry=true&
entry=<Entry Structure="TestForm2"><EntryItem ID="2">$USER$
</EntryItem><EntryItem ID="7">Assigned</EntryItem><EntryItem ID="8">Description</EntryItem></Entry>*
This will return and error since the implicit and explicit structures are different.

Delete

Call: /klink/delete/<Datasource>

Removes an entry from the database.

Sample1: *http://<klinkhost>/klink/delete/<datasource>/<structure>/000000000000001*
This will physically remove the entry with ID 000000000000001 from the structure specified.

Entries

Call: /klink/entries/<Datasource>/<Structure>
Parameters: items, limit, qualification, range, sort, target

This returns a list of entries matching specific criteria. The call retrieves a list of up to the limit specified entries (and any entry items included in the items parameter, or all of them if "all" is specified) matching the qualification, sorts them according to the sort parameter (the entry item id by default), applies the range to this sorted list, then selects a target from that range. The items parameter affects what information about the entry is returned, by default only the entry id. The limit parameter sets the maximum number of entries to retrieve, however the limit can't be larger then that supported by the datasource. The limit defaults to the datasources set limit, or unlimited if the datasource doesn't have a limit. The qualification can be any qualification or query string, which the datasource supports, and defaults to a qualification, which is always true. The range parameter accepts a list of ranges, which can be either single indexes or a range in the format <first>-<last> and defaults to the entire list of entries. The sort parameter takes a list of entry item IDs to sort on and can either use increasing sorting (+) or decreasing sorting (-). By default, entries are sorted by ID. Finally, the target parameter defines which of the entries within the range are to be returned; the first, the last, a random entry, or all of the entries by default.

Sample1: *http://<klinkhost>/klink/entries/<datasource>/TestForm*
This will return the IDs of all of the entries in the TestForm structure.

Sample2: *http://<klinkhost>/klink/entries/<datasource>/BensForm?*
items=7,8&qualification='1'>9&range=1,3-5&sort=1-,7&target=rand
This will retrieve a list of entries with entry item 1 (or Field 1 for Remedy, Request ID by default) greater than nine, sort them based on decreasing values of entry item 1 and increasing values of entry item 7 (Field 7, or status, for Remedy), pick a random entry from the first, third, fourth, or fifth entries, and return the Entry ID, Status (field 7), and Short Description (field 8).

Entry

Call: /klink/entry/<Datasource>/<Structure>/<EntryID>[,<EntryID>[…]]
Parameters: items

This will return the generic xml representation for each of the specified entry IDs. By default, only the entry items that have non-null values will be returned. Null valued entry items can be included by setting the Http parameter items to "all". Additionally, a subset of specific entry items can be requested by passing in a list of entry item IDs through the items parameter.

Sample1: *http://<klinkhost>/klink/entry/<datasource>/<structure>/000000000000001*
This will return an Xml description of the entry with the ID specified from the structure specified (but won't include entry items with null values).

Sample2: *http://<klinkhost>/klink/entry/<datasource>/<structure>/000000000000001 ?items=all*
This will return an Xml description of the entry with the ID specified from the structure specified including all null valued entry items..

Sample3: *http://<klinkhost>/klink/entry/<datasource>/TestForm/000000000000008, 000000000000012*
This will return an Xml description of the entries specified from TestForm (but won't include fields with null values).

Sample4: *http://<klinkhost>/klink/entry/<datasource/TestForm/000000000000008, 000000000000012?items=8,536880920*
This will return an Xml description of the entries specified from TestForm but will only include the entry items with IDs of 8 (short description) and 536880920 (say for example an attachment field).

Update

Call: /klink/create/<Datasource>[[/<Structure>]/<EntryID>]
Parameters: entry, returnentry

This will update an entry within the specified structure. As with create, the entry xml can be specified either by the http parameter "entry" or as an application/xml Http request body and is the same format as that which is returned by the entry framework call. By default, this call will only return an empty Entry element with the ID and Structure attributes set. If a full view of the EntryItems is required, the "returnentry" parameter can be set to true. The Structure can either be set as an attribute to the Entry element or as part of the request Url. If it appears in both and is different, Klink will return a ModelException. Similarly, the entry ID can either be set as an attribute to the Entry element or as part of the request Url. If the entry ID is not present in either, or it appears in both and is different, a ModelException will be thrown.

APPENDIX A SAMPLE CODE

SAMPLE JAVA CALL

```
// Set up the required parameters
String klinkServer = “MyServer:8080”;
String remedyServer = “RemedyServer”;
String credentials = “Demo:demopass”;

// Make the initial connection
String urlString = “http://” + klinkServer + “/klink/create/” + remedyServer;
java.net.URL url = new URL("http://flounder:8080/klink/create/womprat");
java.net.URLConnection conn = url.openConnection();
conn.setDoOutput(true);

// Set the required header information
String encoding = new sun.misc.BASE64Encoder().encode(credentials.getBytes());
conn.setRequestProperty("Authorization", "Basic " + encoding);
conn.setRequestProperty("Accept", "application/xml");
conn.setRequestProperty("Content-Type", "application/xml");

// Write the request
java.io.OutputStreamWriter wr = new OutputStreamWriter(conn.getOutputStream());
wr.write("<Entry Structure=\"TestForm\"/>");
wr.flush();

// Prepare for the response
java.io.InptStreamReader isr = new InputStreamReader(conn.getInputStream());
java.io.BufferedReader rd = new BufferedReader(isr);

// Retrieve the response
String line;
while ((line = rd.readLine()) != null) {
   System.out.println(line);
}

// Clean Up
wr.close();
rd.close();
```