SIMILE Bank APIs

Ryan Lee <ryanlee@w3.org>
For designs by David Huynh <dfhuynh@csail.mit.edu>
November 11, 2005
The SIMILE Project <http://simile.mit.edu/>

A description of each layer of API used throughout SIMILE bank applications.

The following is an introduction to the programming interfaces for user-level commands available in SIMILE bank applications. The SIMILE bank applications - Longwell, on which Piggy Bank and Semantic Bank are built - implement three interfaces layered on top of one another: JavaScript, HTTP, and Java. This document starts at the top of the stack and works its way down to the lower level Java API.

API calls in Longwell are also exposed in Piggy Bank and Semantic Bank; both Banks expose their own additional API calls.

JavaScript

The JavaScript API is a wrapper for the HTTP API, allowing user interfaces the interactivity required to execute parts of the HTTP API at the user's discretion. These calls are dependent on page context provided by the following variables:

`g_contextPath`	where the servlet application is rooted
`g_resourcePath`	where scripts, styles, images, etc. are rooted
`g_profileURL`	the user profile root URL for this page
`g_outerQuery`	the query string used to generate this page
`g_slidingURL`	the URL and non-data related query string fragment of this page
`g_slidingQuery`	the portion of the query string for narrowing down data

The JavaScript functions are all implemented in Longwell and used in Piggy Bank and Semantic Bank. They are grouped under the heading 'operations' and are considered a user interface module. The actual function definitions can be found as client-side script components in the Longwell 2.0 source tree; there is a dependency on the HTTP utility library.

Longwell

`Operations.`**`trust`**`(objectURI, profileID, title)`

`objectURI`	Resource to trust
`profileID`	Profile in which to find the resource
`title`	User alert information

If the identified RDF resource's class is one of several system-defined classes, it will be set to a trusted object. System-defined actions such as downloading extra code or executing commands can be performed according to the trusted object's matching rules. These classes include JavaScript and XSLT screen scrapers.

`Operations.`**`distrust`**`(objectURI, profileID, title)`

`objectURI`	Resource to distrust
`profileID`	Profile in which to find the resource
`title`	User alert information

Removes trust from the previously trusted identified RDF resource.

`Operations.`**`save`**`(objectURI, profileID)`

`objectURI`	Resource to save
`profileID`	Profile in which to find the resource

Saves the identified RDF resource found in temporary models, such as those generated from website metadata, into the locally running bank store.

`Operations.`**`remove`**`(objectURI, profileID)`

`objectURI`	Resource to remove
`profileID`	Profile in which to find the resource

Removes the previously saved identified RDF resource from the locally running bank store.

`Operations.`**`publish`**`(objectURI, profileID)`

`objectURI`	Resource to publish
`profileID`	Profile in which to find the resource

Publishes the identified RDF resource from a locally running bank to all remote banks for all remote bank users to see.

`Operations.`**`persist`**`(objectURI, profileID)`

`objectURI`	Resource to persist
`profileID`	Profile in which to find the resource

Persists the identified RDF resource from a locally running bank to all remote banks for a distributed and private storage.

`Operations.`**`saveAll`**`(url)`

`url`	URL reproducing currently viewed data set for saving

Saves all RDF resources found at currently viewed data set. URL includes pertinent information (profile with currently viewed data, narrowing parameters). Assumes data will be saved to the local profile.

`Operations.`**`removeAll`**`(url, count)`

`url`	URL reproducing currently viewed data set for removal
`count`	User alert information

Removes all data in the currently viewed data set. URL includes pertinent information (profile from which to remove, narrowing parameters).

`Operations.`**`publishAll`**`(publishAllURL, saveAllURL)`

`publishAllURL`	URL reproducing currently viewed data set for publication
`saveAllURL`	URL reproducing currently viewed data set for saving

Publishes all RDF resources found in currently viewed data set and saves all resources before publication if necessary.

`Operations.`**`persistAll`**`(persistAllURL, saveAllURL)`

`persistAllURL`	URL reproducing currently viewed data set for persistence

`saveAllURL`	URL reproducing currently viewed data set for saving

Persists all RDF resources found in currently viewed data set and saves all resources before persistence if necessary.

HTTP

The HTTP API is a web accessible interface to a data repository, currently written in Java as a set of various POST methods. In Semantic Bank, HTTP 'commands' for creating bank accounts, deleting data, and uploading data are exposed to users. In a local bank such as Piggy Bank, HTTP 'commands' are provided for saving web data, publishing and persisting data to a semantic bank, deleting local data, and enabling or disabling data harvesters. The HTTP API for Piggy Bank is of little use to external applications since the actions cannot be decoupled from a user's browsing experience.

An HTTP command specification looks like `method url [parameter(s)]`. See RFC2616 for more on the HTTP POST method [1]. This API treats the parameters part of the command as newline-separated values of the POST payload.

Piggy Bank

Assume the Piggy Bank application is mounted at the URL fragment `/piggy-bank`

POST /piggy-bank/*profile*?command=system [trust] [objectURI]

`profile`	Either 'default' or an ephemeral model
`trust`	Takes either 'trust' or 'distrust' as a value
`objectURI`	Set trust level for this identified resource

Called by `Operations.trust` and `Operations.distrust`.

POST /piggy-bank/*profile*?command=save [objectURI]

`profile`	An ephemeral model
`objectURI`	Save this identified resource

Called by `Operations.save`.

`POST /piggy-bank/default?command=remove [objectURI]`

`objectURI`	Remove this identified resource

Called by `Operations.remove`.

`POST /piggy-bank/`*`profile`*`?command=publish [objectURI]`

`profile`	Either 'default' or an ephemeral model
`objectURI`	Publish this identified resource

Called by `Operations.publish`.

`POST /piggy-bank/`*`profile`*`?command=persist [objectURI]`

`profile`	Either 'default' or an ephemeral model
`objectURI`	Persist this identified resource

Called by `Operations.persist`.

`POST /piggy-bank/`*`profile`*`?command=saveAll [slidingQuery]`

`profile`	An ephemeral model
`slidingQuery`	Query used to generate currently viewed data set

Called by `Operations.saveAll`. This could also be a GET operation.

`POST /piggy-bank/default?command=removeAll [slidingQuery]`

`slidingQuery`	Query used to generate currently viewed data set

Called by `Operations.removeAll`. This could also be a GET operation.

`POST /piggy-bank/`*`profile`*`?command=publishAll [slidingQuery]`

`profile`	Either 'default' or an ephemeral model
`slidingQuery`	Query used to generate currently viewed data set

Called by `Operations.publishAll`. This could also be a GET operation.

`POST /piggy-bank/`*`profile`*`?command=persistAll [slidingQuery]`

`profile`	Either 'default' or an ephemeral model
`slidingQuery`	Query used to generate currently viewed data set

Called by `Operations.persistAll`. This could also be a GET operation

Semantic Bank

Assume the Semantic Bank application is mounted at the URL fragment `/semantic-bank` Note in examples that necessary but variable bits dependent on the client have been elided; only the essential parts required specifically for Semantic Bank operation are included.

`POST /semantic-bank/profile?command=create`

`profile`	Username for new user

Called by native plugin code communicating with remote machine running Semantic Bank. Example:

```
POST /semantic-bank/foo?command=create HTTP/1.1
Host: bank.example.org
Content-Length: 0
```

`POST /semantic-bank/profile?command=upload&format=[format] [data]`

`profile`	Username of user for whom to add data
`format`	`rdfxml`, the serialization of `data`; no other options available
`data`	Serialization of the graph to publish to the bank

Called by remote `Operations.publish, persist, publishAll, persistAll`. Example:

```
POST /semantic-bank/foo?command=upload&format=rdfxml HTTP/1.1
Host: bank.example.org
Content-Length: 317

<rdf:RDF xmlns:rdf=”http://www.w3.org/1999/02/22-rdf-syntax-ns#”
         xmlns:rdfs=”http://www.w3.org/2000/01/rdf-schema#”>
 <rdf:Description rdf:about=”http://www.example.org/ns#item”>
  <rdfs:label>An Example</rdfs:label>
  <rdf:type rdf:resource=”http://www.example.org/ns#Thing”/>
 </rdf:Description>
</rdf:RDF>
```

`POST /semantic-bank/profile?command=remove [objectURI]`

`profile`	Username of user from whom to remove data
`objectURI`	Delete this identified resource

Called by remote `Operations.remove`. Example:

```
POST /semantic-bank/foo?command=remove HTTP/1.1
Host: bank.example.org
Content-Length: 30

http://www.example.org/ns#item
```

Java

The Java API is the lowest level interface available. Servlets interpreting HTTP API commands ultimately make calls to the Java API. In the Longwell architecture, commands are implementations of the `IFlair-Command` interface, which has one method, `execute(FlairMessage msg)`. The `FlairMessage` object attributes:

`FlairServlet m_servlet`	Flair servlet, controls commands
`HttpServletRequest m_request`	Raw servlet request
`HttpServletResponse m_response`	Raw servlet response
`VelocityEngine m_ve`	Rendering engine for generating returned page
`String m_profileID`	The username associated with a profile
`Query m_query`	The query string
`String m_locale`	User locale (for string localization purposes)

The `FlairMessage` object methods:

`FlairMessage(FlairServlet servlet, HttpServletRequest request, HttpServletResponse response, VelocityEngine ve, String profileID, Query query, String locale)`	Creates a new object with the arguments as attributes
`Profile getProfile()`	Returns the user profile given the `m_profileID`
`Query getQuery()`	Returns the query string, `m_query`
`String getURL(Value v)`	Converts known URIs to focus-type bank URLs, tries to convert literals to URLs

`IFlairCommand`'s are mostly composed of manipulations of information derived from the `FlairMes-sage`. It is recommended that developers build on the existing Longwell application. One can examine the source for each implementation of `IFlairCommand` for a better idea of what actual actions are taken when the JavaScript methods are executed or data posted to URLs.

Piggy Bank

Assume the Piggy Bank application is mounted at the URL fragment `/piggy-bank`

`edu.mit.simile.piggyBank.servlet.PersistAllCommand`
Called by /piggy-bank/profile?command=persistAll

`edu.mit.simile.piggyBank.servlet.PersistCommand`
Called by /piggy-bank/profile?command=persist

`edu.mit.simile.piggyBank.servlet.PublishAllCommand`
Called by /piggy-bank/profile?command=publishAll

`edu.mit.simile.piggyBank.servlet.PublishCommand`
Called by /piggy-bank/profile?command=publish

`edu.mit.simile.piggyBank.servlet.RemoveAllCommand`
Called by /piggy-bank/default?command=removeAll

`edu.mit.simile.piggyBank.servlet.RemoveCommand`
Called by /piggy-bank/default?command=remove

`edu.mit.simile.piggyBank.servlet.SaveAllCommand`
Called by /piggy-bank/profile?command=saveAll

`edu.mit.simile.piggyBank.servlet.SaveCommand`
Called by /piggy-bank/profile?command=save

`edu.mit.simile.piggyBank.servlet.SystemCommand`
Called by /piggy-bank/profile?command=system

Semantic Bank

Assume the Semantic Bank application is mounted at the URL fragment `/semantic-bank`

`edu.mit.simile.semanticBank.servlet.CreateAccountCommand`
Called by /semantic-bank/profile?command=create

`edu.mit.simile.semanticBank.servlet.RemoveCommand`
Called by /semantic-bank/profile?command=remove

`edu.mit.simile.semanticBank.servlet.UploadCommand`
Called by /semantic-bank/profile?command=upload

References

1. Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L. Leach, P., Berners-Lee, T. *HyperText Transfer Protocol -- HTTP/1.1* Request for Comments 2616, Network Working Group `<http://rfc.net/rfc2616.html>`, Jun. 1999.