ActiveRaUL User Guide

# Overview

The ActiveRaUL system consists of two main parts: (1) a RESTful Web service that provides a uniform way to manage RaUL-based Web forms and data that is processed by those Web forms; (2) a client-side JavaScript RDFa API based on jQuery that enables the developer to interact with the ActiveRaUL web service.

This document provides you the instruction to install the web service component of the ActiveRaUL system on your server and the overview of the client-side JavaScript API.

# Prerequisites

* Java 5 or newer
* Apache Ant
* A Java Servlet Container that supports Java Servlet API 2.4 and Java Server Pages (JSP) 2.0, or newer.
* Sesame 2 RDF Store installed

At the time of writing, the ActiveRaUL system has been tested with Java version "1.7.0", Ant version "1.8.2", Tomcat 6 and Sesame 2.6.3.

# Installation under Tomcat

The easiest way to install the ActiveRaUL service on Tomcat is using Tomcat Web Application Manager to upload the "raul.war" file. Otherwise, you can manually deploy the ActiveRaUL service on Tomcat according to the following steps.

1. Build the ActiveRaUL service from source by typing "ant clean; ant dist" under the directory that contains the ActiveRaUL source code.
2. Create a native repository with RDF Schema inferencing on Sesame server, and designate the newly create repository as "raul-default". (Please refer to the documents of Sesame server for how to create an RDF repository.)
3. Create a "raul" directory under the web applications directory ([TOMCAT\_HOME]/webapps/ by default).
4. Extract the "raul.war" file to the newly created "raul" directory.
5. Restart your Tomcat server and the ActiveRaUL service should now be up and running.

Note that, you could change the configuration of the RDF repository setting by editing "raul.properties" under the "/src/main/java" directory. Having done so, you should reload the ActiveRaUL service to activate the change.

# Test

You can point a web browser to the location where you have installed Sesame (e.g. http://[MACHINE\_NAME]:8080/raul/ if you have installed the ActiveRaUL service under Tomcat) to test your installation. The browser should now display a demo web site of the ActiveRaUL system.

You could also test whether the ActiveRaUL service works correctly by running one of demo applications. For example, the "FOAF Editor" demo (e.g. <http://[MACHINE_NAME]:8080/raul/foafedit.html>) replicates the functionality provided by FOAF-a-matic (see <http://www.ldodds.com/foaf/foaf-a-matic>). Instead of developing a specific Web page for a specific ontology (i.e. FOAF), we create a RaUL RDF model to create, retrieve and update the RDF graph describing a Person. This Graph is then deployed with the ActiveRaUL service (by clicking the "Form Creation" button) which, in turn, can render a Web form in XHTML+RDFa. Moreover, the resulting Web form and FOAF profile can be reused via their URIs. Of course, you could fill out the FOAF editor form and submit a data instance to the ActiveRaUL service. If data about a person already exists somewhere else on the Web, you could provides a URI reference to an existing file in the "Keyword(s)/FOAF URI" field, or type in a search term that is used to query the Linked Data Web for appropriate data. Having done so, the data will be filled in the form. The ActiveRaUL system has been successfully installed if all of this works.

# Web Service Interface

|  |  |  |
| --- | --- | --- |
| CRUD Operation | HTTP Request | Web Service Path |
| Create a form | POST | /{userid}/forms |
| Create a data instance | POST | /{userid}/forms/{formid} |
| Read an existing form | GET | /{userid}/forms/{formid} |
| Read a data instance | GET | /{userid}/forms/{formid}/{dataid} |
| Update a form | PUT | /{userid}/forms/{formid} |
| Update an data instance | PUT | /{userid}/forms/{formid}/{dataid} |
| Delete an existing form | DELETE | /{userid}/forms/{formid} |
| Delete an existing data instance | DELETE | /{userid}/forms/{formid}/{dataid} |

A sophisticated user authentication mechanism has not been provided in the current version of the ActiveRaUL system; however, you could extend the class "UserManager" (in "/src/main/java/ie/deri/raul/UserManager.java") to implement your own user authentication.

# Client-side JavaScript API

#### function initRaulFrontEnd()

For client-side JavaScript API initialization, call this function before using the API.

#### function postForm(formDef)

For creating a RaUL-based form (post a form definition).

* Parameters:
  + *formDef* -- the form definition described in RDF/XML format (string type).
* Retrun values: the URI of the deployed form (assigned by the ActiveRaUL service).

#### function getForm(formURI)

For retrieving an exist form.

* Parameters:
  + *formURI* -- the form URI.
* Return value: the form definition in XHTML/RDFa format.

#### function postData(formURI, data)

For submitting user input data to the server.

* Parameters:
  + *formURI* -- the form URI.
  + *userInputData* -- the user input data.
* Return value: *t*he URI of the submitted data instance (assigned by ActiveRaUL service).

#### function getData(dataURI)

For retrieving an existing data instance.

* Parameters:
  + *dataURI* -- the data instance URI.
* Return value: *t*he request data in XHTML/RDFa format.

#### function deleteInstance(instanceURI)

For deleting an existing form/data instance.

* Parameters:
  + instanceURI -- the form/data URI.

#### function putForm(formURI, formDef)

For updating an existing form.

* Parameters:
  + *formURI* – the URI of the form that you want to update.
  + *formDef* -- the form definition described in RDF/XML format (string type).

#### function putData(dataInstanceURI, userInputData)

For updating an existing data instance.

* Parameters:
  + *dataInstanceURI* -- the URI of the data instance that you want to update.
  + *userInputData* -- the user input data.

#### function getRequestRDF(requestURI)

For retrieving the RDF graph of an existing form/data instance.

* Parameters:
  + *requestURI* -- the form/data URI.