Linked Widgets Platform - Documentation

1. Introduction to Linked Widget Platform

Linked Widgets Platform (LWP) is a framework for exploring and integrating heterogeneous data sources in a reusable and flexible manner. It focuses on high-level data processing and acts as a problem-oriented mashup system.

In the platform, we modularize functionality into **Linked Widgets** so that end users can recombine and create complete applications. To this end, Linked Widgets have input and/or output terminals. Connecting an input terminal of a widget with an output terminal of another means the former widget accepts and processes the output data of the latter as its input data. We use JSON-LD for data transmission between widgets.

The platform follows the **open principle**. It allows anyone to develop and contribute their own widgets. Those widgets can be deployed on arbitrary servers.

Depending on their execution mode, widgets may be classified as **client** or **server (linked) widgets**. From a functional point of view, we further divide Linked Widgets into three categories, i.e., ***data***, ***processing***, and ***visualization*** widgets which perform the data retrieval, data processing/data integration, and data presentation tasks, respectively.

Two mandatory components for a complete application are data and visualization Linked Widgets; processing widgets are optional. A data widget does not have any input terminals; it collects data from an arbitrary data source to provide input for processing and visualization widgets. Visualization widgets display output data of processing or data widgets. The specification for number of input and output terminals for three types of widget is shown in Table 1.

Table 1 Specification for number of terminals for each type of widget

Widget Type	Number of Input Terminals	Number of Output Terminals
Data widget	0	1
Processing widget	1 or many	1
Visualization widget	1 or many	0

2. Client Widget Developments

2.1 General guidelines

To develop a client Linked Widget, we use the template shown in <Figure 1> as follows:

1. Update the title of the HTML file
2. Update the widget configuration, where we can specify the size of widget as well as its input and/or output terminals. The widget described in the template is a processing widget with two input terminals (which are “input1” and “input2”) and one output terminal. We need to define relative left and top positions of each terminal to the widget.
3. Implement the run function, which receives input data from preceding widgets and returns the processed output data to the succeeding widgets
4. Implement the widget interface

```
<!DOCTYPE html>
<html>
<head>
  <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
  <title>Your widget title</title>
  <script type="text/javascript" src="http://linkedwidgets.org/widgets/WidgetHub.js"></script>
  <script type="text/javascript" src="http://linkedwidgets.org/autobahn.min.jgz"></script>
  <script type="text/javascript">
    var config = {
      size: {width: 400, height: 250}, // the width, height of our widget (in pixels)
      terminals: [ // define the position of our terminals
        {"name": "input1", "position" : {"left": 0, "top": 1/3}, "type": "input"},
        {"name": "input2", "position" : {"left": 0, "top": 2/3}, "type": "input"},
        {"name": "output", "position" : {"left": 1, "top": 1/2}, "type": "output"}
      ],
      resizable: true // decide whether our widget is resizable or not
    };

    /*
    This function defines the job of our widget.
    */
    function run(input) {
      var inputObject1 = input["input1"]; // data received at the first input terminal
      var inputObject2 = input["input2"]; // data received at the second input terminal
      var outputObject = {}; // Compose output data object based on inputObject1and inputObject2
      hub.returnOutput(outputObject); // When output data is ready, use this statement
    }

    var hub = new WidgetHub(config, run);
  </script>
</head>
<body>
Build your widget interface here
</body>
</html>
```

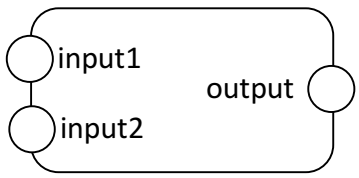

Figure 1 Client Linked Widget template. The red code should be updated for specific widgets.

2.2 Simple widget collection

Let's introduce a simple collection of three widgets as follows:

1. US President – a data widget that returns a list of US presidents with their name and birth year <Figure 2>
2. Person Filter – a processing widget that filters a list of persons based on their birth year <Figure 3>
3. Person Viewer – a visualization widget that visualizes a list of persons <Figure 4>

```
<!DOCTYPE html>
<html>
<head>
  <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
  <title>US President</title>
  <script type="text/javascript" src="http://linkedwidgets.org/widgets/WidgetHub.js"></script>
  <script type="text/javascript" src="http://linkedwidgets.org/autobahn.min.jgz"></script>
  <script type="text/javascript">
    var config = {
      size: {width: 150, height: 80}, // the width, height of our widget (in pixels)
      terminals: [ // define the position of our terminals
        {"name": "output", "position" : {"left": 1, "top": 1/2}, "type": "output"}
      ],
      resizable: true // decide whether our widget is resizable or not
    };
    function run(input) { // This is a data widget so we just ignore the input object
      var outputObject = {
                  "@context": {
                    "name": "http://xmlns.com/foaf/0.1/name",
                    "birthYear": "http://dbpedia.org/ontology/birthYear"
                  },
                  "@graph": [
                    {
                      "@id": "http://dbpedia.org/resource/Barack_Obama",
                      "@type": "http://xmlns.com/foaf/0.1/Person",
                      "name": "Barack Obama",
                      "birthYear": 1961
                    },
                    {
                      "@id": "http://dbpedia.org/resource/Donald_Trump",
                      "@type": "http://xmlns.com/foaf/0.1/Person",
                      "name": "Donald Trump",
                      "birthYear": 1946
                    }
                  ]
      };
      hub.returnOutput(outputObject); // return output data
    }
    var hub = new WidgetHub(config, run);
  </script>
</head>
<body>US Presidents</body>
</html>
```

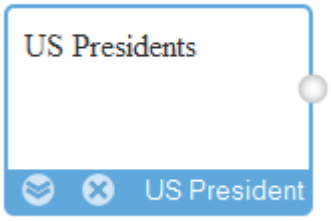

Figure 2 The "US President" widget that generates a list of US presidents. This is a data widget and has a single output terminal only.

```
<!DOCTYPE html>
<html>
<head>
  <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
  <title>Person Filter</title>
  <script type="text/javascript" src="http://linkedwidgets.org/widgets/WidgetHub.js"></script>
  <script type="text/javascript" src="http://linkedwidgets.org/autobahn.min.jgz"></script>
  <script src="http://code.jquery.com/jquery-latest.min.js" type="text/javascript"></script>
  <script type="text/javascript">
    var config = {
      size: {width: 300, height: 80}, // the width, height of our widget (in pixels)
      terminals: [ // define the position of our terminals
        {"name": "input", "position" : {"left": 0, "top": 1/2}, "type": "input"},
        {"name": "output", "position" : {"left": 1, "top": 1/2}, "type": "output"}
      ],
      resizable: true // decide whether our widget is resizable or not
    };

    function run(input) {
      var inputObject = input["input"];

      if ($('#birthYear').val()!=""){
        var birthYear = parseInt($('#birthYear').val());
        for(var i=inputObject["@graph"].length -1; i>=0; --i){
          if (inputObject["@graph"][i]["birthYear"] < birthYear) {
            inputObject["@graph"].splice(i, 1);
          }
        }
      }

      var outputObject = inputObject;
      hub.returnOutput(outputObject); // When output data ready, use this statement
    }

    var hub = new WidgetHub(config, run);
  </script>
</head>
<body>
<label for="birthYear">Minimum birth year: </label><input type="text" id="birthYear"/><br/>
</body>
</html>
```

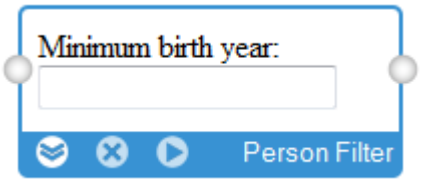

Figure 3 The "Person Filter" widget, which is a processing widget. It has a single input and output terminal. It receives a list of persons and removes some that do not meet the filtering condition.

```
<!DOCTYPE html>
<html>
<head>
  <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
  <title>Person Viewer</title>
  <script type="text/javascript" src="http://linkedwidgets.org/widgets/WidgetHub.js"></script>
  <script type="text/javascript" src="http://linkedwidgets.org/autobahn.min.jgz"></script>
  <script src="http://code.jquery.com/jquery-latest.min.js" type="text/javascript"></script>
  <script type="text/javascript">
    var config = {
      size: {width: 400, height: 250}, // the width, height of our widget (in pixels)
      terminals: [ // define the position of our terminals
        {"name": "my_single_input", "position" : {"left": 0, "top": 1/2}, "type": "input"}
      ],
      resizable: true // decide whether our widget is resizable or not
    };

    function run(input) {
      $("#mycontent").empty();
      var inputObject = input["my_single_input"];

      for(var i=0; i<inputObject["@graph"].length; ++i){
        var birthYear = inputObject["@graph"][i]["birthYear"];
        var name = inputObject["@graph"][i]["name"]
        $("#mycontent").append("<p>" + (i+1) + ". " + name + " - " + birthYear + "</p>");
      }

      hub.returnOutput(inputObject); // Visualization widget should return the inputObject
    }

    var hub = new WidgetHub(config, run);
  </script>
</head>
<body>
  <div id="mycontent"></div>
</body>
</html>
```

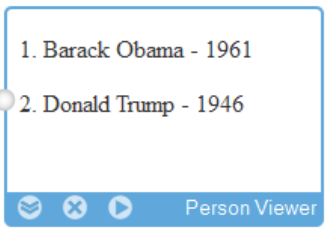

Figure 4 The "Person Viewer" widget that visualizes a list of persons.

2.3 Develop your widget collections

The best method to develop, debug, and test your Linked Widgets is to do so at your local computer. To this end, you need to install such web servers as IIS, XAMPP or Apache Tomcat. Select the server you are most familiar with, create a web application, say, ***mywidgets***, and develop your widgets. For example, the links to the three widgets can be as follows:

http://localhost:8080/mywidgets/uspresident/index.html
http://localhost:8080/mywidgets/personfilter/index.html
http://localhost:8080/mywidgets/personviewer/index.html

Next, open your web browser and use the http://linkedwidgets.org/ (or if you already installed LWP in your localhost, you can open http://localhost:8080/lw-client/) URL to go to the Linked Widgets platform. At the bottom of the home page, you see a carousel panel. You will add the name and the URL of each widget to this carousel as shown in <Figure 5>.

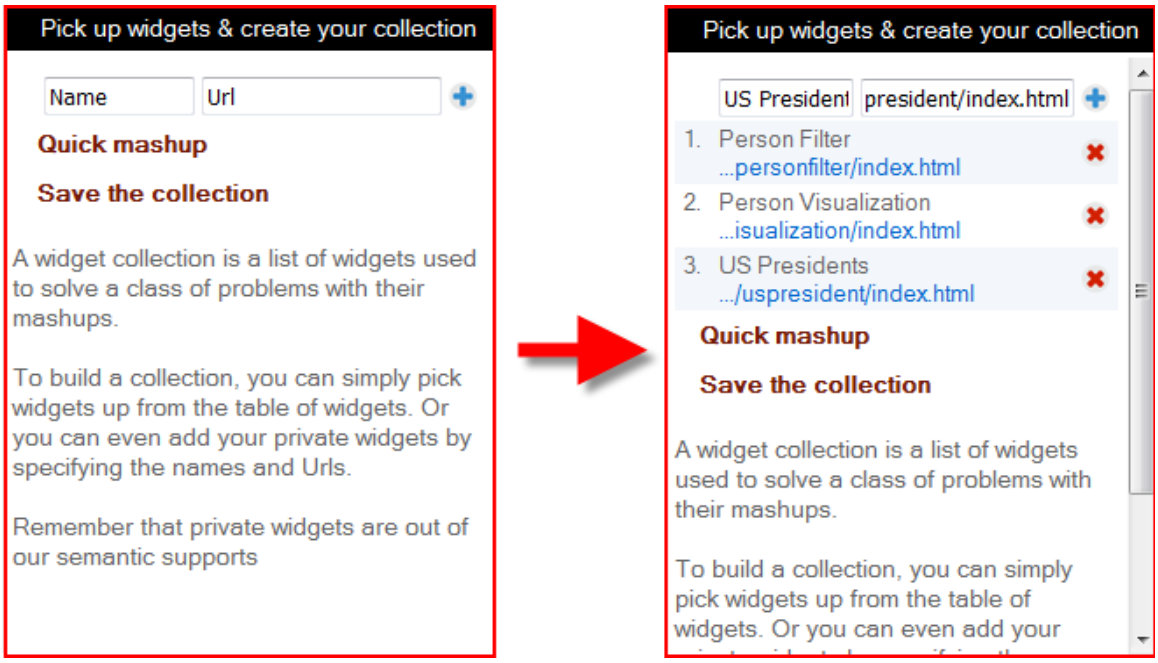

Figure 5 Create a widget collection

Now, press the quick mashup button to go to the mashup editor panel. You then drag, drop, and combine widgets to create a mashup as shown in **Figure 6**

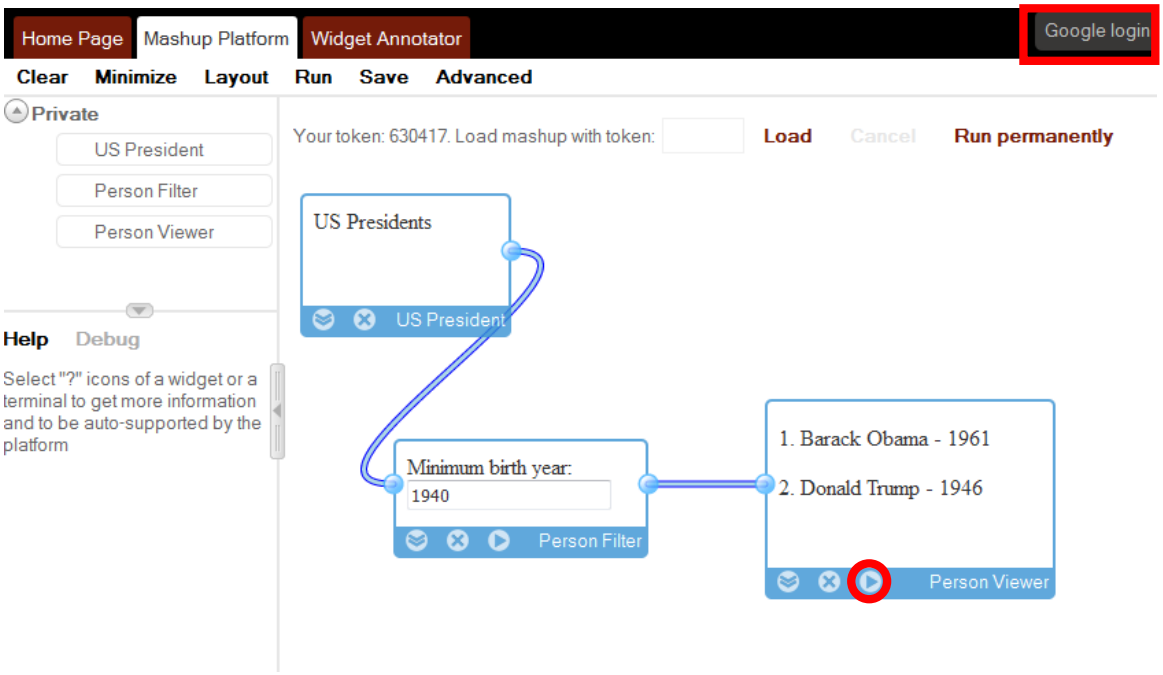

Figure 6 Sample mashup. Press the run button of the visualization widget to execute the mashup.

To debug the data of an output terminal **O** of any widget, please use the JSON Viewer widget. An instance of this widget can be added to the mashup editor when you (1) hover this output terminal **O**, (2) select the question mark, and (3) select the JSON Viewer widget on the help panel (cf. **Figure 7**).

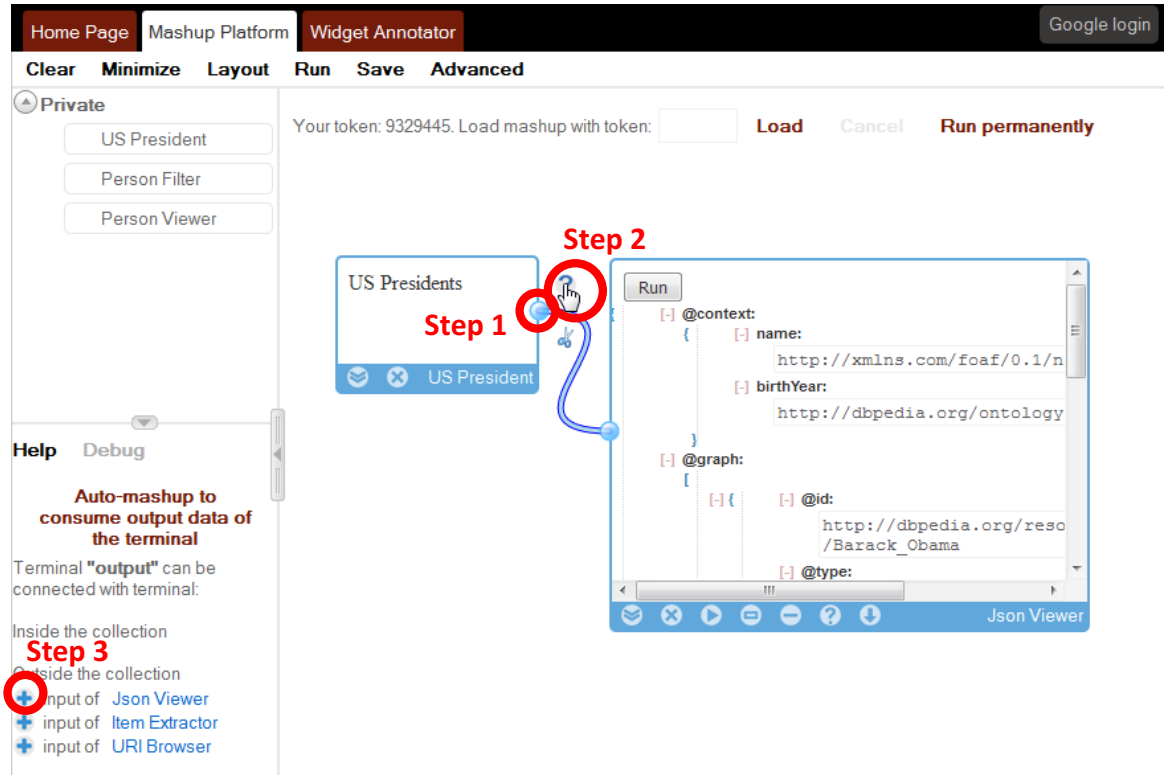

Figure 7 Add JSON Viewer widget to debug the output of a widget

2.4 Adding client widgets in the example-widgets project

As an alternative to directly add your client widgets in the LWP (as shown in Section 2.1-2.3), you can also add your client widgets as part of the example-widgets project, using the following steps:

- Create your client widget's HTML file that contain necessary javascript.
- Register your client widget
    - In the example-widgets project, we wrap the registration of client widgets within ClientWidgetRegistry class (src/main/java/org/linkedwidgets/example/widget/client/ClientWidgetRegistry.java)
    - In this class, you need to add another call of InitializeWidget() function with the following parameters
        - name: the name of your widget
        - widgetType: "data", "process", or "visualization"
        - htmlURL: the url of your widget html file.
        - description: the description of your widget
        - isExternal: it is true if your client widget's html file is not within the example-widgets project. In this case, you should add the full url address of the html file within the parameter htmlUrl.
- If everything goes correctly, after rebuilding the project and redeploying the example-widgets.war, you should see your new client widget(s) in the LWP
- You can see several examples of client widget (in the "src/main/webapp/html/client/" folder) and their InitializeWidget() call (in the ClientWidgetRegistry.java class).

3. Server Widget Developments

In order to create a Java server widget and add it into the example-widgets project, you will need to do the following steps:

- Create an HTML file containing necessary JavaScript
- Create a Java class that extends ServerWidgetJob class, and overwrite the following two functions:
    - run() function to allow your server widget to receive inputs and parameters, do some processing (e.g., loading, merge, etc.) and produce an output
    - destroy() function to allow you cleaning up your application in order to avoid memory leak when the run function finished.
- Instantiate and register your ServerWidgetJob class into LWP.
    - In the example-widgets project, we wrap the instantiation and registration of server widgets within ServerWidgetRegistry class (src/main/java/org/linkedwidgets/example/widget/server/ServerWidgetRegistry.java)
    - After you finish creating your ServerWidgetJob class extension, you will need to instantiate your widget by adding another call of initializeWidget() function with the following parameters
        - widgetJob: an instance of your widgetJob class
        - widgetName: the name of your widget
        - widgetType: type of your widget, can be "data" or "process"
        - htmlURL: the link to your widget html file. Please put your html file inside "src/main/webapp/html/server/" folder
        - description: the description of your widget
        - widgetId: your widget id. the id should be in lower case and should not contain any space or special characters
- If everything goes correctly, after rebuilding the project and redeploying the example-widgets.war, you should see your new server widget(s) in the LWP
- You can see several examples of the ServerWidgetJob class extension (in the “src/main/java/org/linkedwidgets/example/widget/server/” folder), their respective html file (in the "src/main/webapp/html/server/" folder) and their instantiations (in the ServerWidgetRegistry.java class).

.