Client Project Lookup Javascript Bridge 1.0 Component Specification

1. Design

This component provides an AJAX bridge, allowing JavaScript components or web-pages to interact with the Client Project Lookup Services 1.0 component. The component has two distinct parts. The first part is a set of JavaScript functionality which provides an API mirroring that of the Client Project Lookup Services component. This API interacts with the other part, a Java servlet, through AJAX requests. The servlet translates the requests into parameters which are used to call into the Client Project Lookup Services component, and then converts returned values into an AJAX response which is returned to the JavaScript caller.

1.1 Design Patterns

Strategy Pattern: the ClientProjectLookupServiceBridgeServlet uses JSONEncoder, JSONDecoder and Log as strategies.

Proxy Pattern: the LookupService, ClientStatusService, ProjectStatusService are corresponding proxies of the web services deployed in server.

1.2 Industry Standards

AJAX, JSON, Servlet

1.3 Required Algorithms

1.3.1 Logging

The ClientProjectLookupServiceBridgeServlet should log all calls at DEBUG level, including the name of the service ("lookup", "clientStatus", or "projectStatus") and name of service method which is to be called, and information to identify the objects involved - object ids but not the complete parameter data.

And all thrown exceptions should be logged with WARNING level, including the exception message and exception stack trace.

NOTE: Logging is only done in the doPost and doGet method.

1.3.2 Validate Argument Type in JavaScript Functions

The arguments passed into the JavaScript Functions should be verified to see if they have expected data types. Throw IllegalArgumentException if not.

1.3.3 Evaluate JSON string in JavaScript

We can simply call: `var jsonObj = eval("(" + jsonText + ")")` to evaluate the jsonText (string value) into a JSON object.

Developer may also choose to use `JSON.parse` to evaluate the JSON string, but it's not required as the security is not a concern currently.

1.3.4 Date Precision

When converting the Date value into JSON string, the hours and minutes should be included as well as date.

1.3.5 JavaScript Service

The JavaScript services (LookupService, ClientStatusService and ProjectStatusService) will send AJAX requests to a configured servlet url, and the returned response will be JSON object in string representation.

1.3.5.1 Success Response

If the request is handled successfully on the server side, the returned JSON object will be:
`{ "success" : true, "json" : $json }`

Where the “success” property indicates the operation succeeds, and the “json” property represents the JSON object corresponding to the returned value of specific web service method. The “json” property is not present if the corresponding web service method doesn’t return anything.

And after parsing the returned JSON response, the onSuccess callback function will be called.

NOTE: The corresponding web service method is likely to return a null value, in this case, the $json value will be set to an empty string to indicate it. And the onSuccess callback function should be called with a null value.

1.3.5.2 Failure Response

If error occurs on the server side when processing the request, the returned JSON object will be: `{ "success" : false, "error" : $error-message }`

Where the “success” property indicates the operation fails, and the “error” property indicates the error message.

And after parsing the returned JSON response, the onError callback function will be called.

1.3.6 Handle the request in ClientProjectLookupServiceBridgeServlet

The requests sent by the JavaScript Services Classes will always have the “service” and “method” parameters.

The “service” parameter value can be “lookup”, “clientStatus” and “projectStatus”, and they correspond to the LookupService, ClientStatusService, and ProjectStatusService web services respectively.

The “method” parameter value represents the method name of the corresponding web service. For example, if the “service” parameter value is “lookup”, and the “method” parameter value is “getClient”, then the getClient method of the LookupService should be called to send the request to the web service.

When calling web service, first the client’s getXXXPort() method should be used to get web service endpoint, then use the endpoint to invoke web service.

And depending on the web service method to call, we should also extract corresponding method arguments’ values from the request parameters.

Take the LookupService’s getClientsForCompany method as an example:
1. The request contains the “company” parameter, whose value is a JSON string.
2. Call the jsonDecoder.decodeObject to decode the JSON string into a JSONObject.
3. Then convert the created JSONObject into a Project Java object.

4. Call lookupService.getLookupServicePort().getClientsForCompany method with the Company object created above.
5. Get the returned value, and convert it into a JSONObject.
6. Call the jsonEncoder.encode to encode the JSONObject created above into a string.
7. If no error occurs, write the following JSON string into response (by calling response.getWriter().print):
   ```
   { “success” : true, “json” : $json }
   ```
   Where $json is generated above.

   If any error occurs (e.g. missing parameter), write the following JSON string into response:
   ```
   { “success” : false, “error” : $error-message }
   ```
   Where $error-message is the exception’s message.

NOTES:
For web service method that doesn’t return any value, we will write:
`{ “success” : true }` into the response if no exception occurs.
If any parameter is missing (if it’s required) or invalid, write the failure response to client.
If the “service” or “method” parameter values are unrecognizable, write failure response to client.
If the web service method returns null value, the $json will be simply set to an empty string.
The IOException will be propagated, and the other exceptions thrown should all be caught and logged, and then write a failure response to client.
If the returned result is just a single entity, the JSON string to send to client is like {property1:value1, property2:value2}.
If the returned result is an array of entities, the JSON string is like
[json1, json2, …].
The searching methods with “filter” argument are not supported, this is confirmed by PM in the forum.

1.3.6.1 Conversion between JSONObject/JSONArray and corresponding Entity Java Object

Please refer to the JSON Object component to see how the JSONObject and JSONArray map to the JSON strings.
This design provides the JSON strings for each JavaScript Object in its toJSON() method, and the JavaScript Objects can map to the corresponding Entity Java Objects easily as their property names are the same. So it should be easy for developers to figure out how the conversion is done.

Here gives an example:

Received JSON string:
```
{"id" : 1, "name" : "topcoder", … }
```

After decoded by jsonDecoder, a JSONObject object is created. (Assume its variable name is jsonClient).

```
// Create a Client entity:
Client client = new Client();

// Populate the client entity with data from the JSONObject.
client.setID(jsonClient.getLong(“id”));
client.setName(jsonClient.getString(“name”));

// And the conversion from client to a JSONObject is similar
JSONObject jsonObj = new JSONObject();
```

```
jsonObj.setLong(“id”, client.getID());
jsonObj.setString(“name”, client.getName());
```

1.3.7 *Send request to ClientProjectLookupServiceBridgeServlet*

The parameters to be posted to the servlet include:
service: The “service” parameter value can be “lookup”, “clientStatus” and “projectStatus”, and they correspond to the LookupService, ClientStatusService, and ProjectStatusService web services respectively.

method: The “method” parameter value represents the method name of the corresponding web service. For example, if the “service” parameter value is “lookup”, and the “method” parameter value is “getClient”, then the getClient method of the LookupService should be called to send the request to the web service.

parameter1: value1
parameter2: value2
parameter3: value3
…

Where parameter1, parameter2, etc. are method argument names; value1, value2, etc. are string representations of the argument values, for type of entities, they are JSON strings.

Here is sample code to send request to servlet to get clients for company, other service methods are similar:

```
// Create AJAXProcessor object
var processor = new AJAXProcessor();
// Send a request asynchronously
processor.request({
    url:  servletUrlString,
    async: true,
    method: "POST",
    // the json string should be escaped properly to a valid URL
    sendingText: "service=lookup&method=getClientsForCompany&company="
        + escape(company.toJSON()),
    onStateChange: function() {
        // Handle the response
        if (processor.getState() == 4 && processor.getStatus() == 200) {
            var response = processor.getResponseText();
            var jsonResp = eval("(" + response + ")");
            // check response
            if (jsonResp == null) {
                throw InvalidResponseException;
            }
            if (typeof(jsonResp.success) == "undefined") {
                throw InvalidResponseException;
            }
            // now check if valid or not
            if (jsonResp.success == false) {
                if (typeof(jsonResp.error) == "undefined") {
                    throw InvalidResponseException;
                }
                // errors
                Call error handler with error message;
            } else {
                if (typeof(jsonResp.json) == "undefined") {
                    throw InvalidResponseException;
                }
                // success
```

```
                    Evaluate jsonResp.json to clients array.
                    Call the success callback passing the clients.
                }
            }
        }
    });
```

1.4 Component Class Overview

1.4.1 Package com.topcoder.clients.bridge.lookup (Java)

- **ClientProjectLookupServiceBridgeServlet**: This class extends the HttpServlet class, and it will decode AJAXrequests from the JavaScript part into parameters used to call Web Service methods in the Client Project Lookup Services component. Returned values should be encoded into the AJAX response.

1.4.2 Package js.topcoder.clients.bridge.lookup.entities (JavaScript)

- **BaseEntity**: This JavaScript object represents the base entity for other entities, all its variables and methods should be defined in subclasses.
- **Company**: This JavaScript object represents the company data.
- **Client**: This JavaScript object represents the client data.
- **Project**: This JavaScript object represents the project data.
- **ClientStatus**: This JavaScript object represents the client status data.
- **ProjectStatus**: This JavaScript object represents the project status data.

1.4.3 Package js.topcoder.clients.bridge.lookup (JavaScript)

- **LookupService**: This JavaScript class defines the operations related to the client project lookup service. It will use the AJAX Processor JS component to communicate with the ClientProjectLookupServiceBridgeServlet. Each method has onSuccess and onError callback functions as its arguments, and as the operation is completed in asynchronous mode, so when the operation is completed successfully, onSuccess will be called, otherwise, onError will be called to notify user an error occurred.
- **ClientStatusService**: This JavaScript class defines the operations related to the client status service. It will use the AJAX Processor JS component to communicate with the ClientProjectLookupServiceBridgeServlet. Each method has onSuccess and onError callback functions as its arguments, and as the operation is completed in asynchronous mode, so when the operation is completed successfully, onSuccess will be called, otherwise, onError will be called to notify user an error occurred.
- **ProjectStatusService**: This JavaScript class defines the operations related to the project status service. It will use the AJAX Processor JS component to communicate with the ClientProjectLookupServiceBridgeServlet. Each method has onSuccess and onError callback functions as its arguments, and as the operation is completed in asynchronous mode, so when the operation is completed successfully, onSuccess will be called, otherwise, onError will be called to notify user an error occurred.
- **CallbackFunctionSignatures**: This entity doesn't corresponds to any *real* JavaScript stuff.

1.5 Component Exception Definitions

1.5.1 System Exceptions (Java)

- **IllegalArgumentException**: It is thrown when the passed-in argument is illegal. NOTE: A string is empty if its length is 0 after being trimmed.
- **IOException**: It is thrown if any I/O error occurs.

1.5.2 Custom Exceptions (JavaScript)

- **IllegalArgumentException**: This exception is thrown if the passed-in argument is invalid. (Refer to method documentation for more details).
- **InvalidResponseException**: It is thrown if the received response is invalid.

1.6 Thread Safety

This component is not completely thread-safe, but it can be used thread-safely.

The ClientProjectLookupServiceBridgeServlet class has mutable variables, but all of them will be initialized once and never changed afterwards. The initialization is done before the servlet is used to serve the user requests, so this class is thread-safe even though it has mutable variables.
The dependent TC components are either thread-safe or used thread-safely.

The JavaScript Entity classes are all mutable and not thread-safe.

The JavaScript Service classes are immutable (user shouldn't change their variables directly) and thread-safe. User can call more than one service methods at the same time, but user should be better not to pass the same JavaScript entity objects to different service methods especially when the entity objects will be changed externally by user, which might cause unexpected result. (NOTE: The JavaScript entity objects passed into JavaScript Service classes will not be changed.)

2. Environment Requirements

2.1 Environment

- Java 1.5+
- Any Servlet Container

2.2 TopCoder Software Components

- **Logging Wrapper 2.0 -** Used to log invocation information and exceptions.
- **AJAX Processor 2.0 -** Used to send AJAX request and receive response.
- **JSON Object 1.0 -** Used to convert data between the JSON string and JSON java objects.
- **Client Project DAO 1.0 -** its entities are used.
- **Client Project Lookup Services 1.0 -** It contains the web services to be called.
- **Configuration API 1.0 -** Used to hold configuration for the servlet.
- **Configuration Persistence 1.0 -** Used to load configuration from persistence.
- **Object Factory 2.0.1 -** Used to create web service wrapper classes.
- **Object Factory Configuration API Plugin 1.0 -** Used to create the Object Factory.

2.3 Third Party Components

None.

3. Installation and Configuration

3.1 Package Name

js.topcoder.clients.bridge.lookup (JavaScript)

js.topcoder.clients.bridge.lookup.entities (JavaScript)

com.topcoder.clients.bridge.lookup (Java)

3.2 Configuration Parameters

For ClientProjectLookupServiceBridgeServlet (web.xml in the docs is a sample config for this servlet):

Parameter Name	Parameter Description	Parameter Value
loggerName	Represents the logger name used to create logger from LogManager. Optional. Default to the full qualified class name of ClientProjectLookupServiceBridgeServlet.	Must be non-empty string if present.
objectFactoryNamespace	Represents the namespace used to get the ConfigurationObject from ConfigurationFileManager to create the ConfigurationObjectSpecificationFactory object. Required	Must be non-empty string.
lookupServiceClientKey	The key used to create the LookupServiceClient object from Object Factory. Optional. Default to “lookupServiceClient”	Must be non-empty string if present.
clientStatusServiceClientKey	The key used to create the ClientStatusServiceClient object from Object Factory. Optional. Default to “clientStatusServiceClient”.	Must be non-empty string if present.
projectStatusServiceClientKey	The key used to create the ProjectStatusServiceClient object from Object Factory. Optional. Default to “projectStatusServiceClient”.	Must be non-empty string if present.
jsonEncoderKey	The key used to create the JSONEncoder object from Object Factory. Optional. Default to “jsonEncoder”.	Must be non-empty string if present.
jsonDecoderKey	The key used to create the JSONDecoder object from Object Factory. Optional. Default to “jsonDecoder”.	Must be non-empty string if present.
ajaxBridgeConfigFile	Represents the configuration file used to create the ConfigurationFileManager object. Required.	Must be non-empty string.

3.3 Dependencies Configuration

Dependent components should be configured according to their component documents.

Especially, the web services of Client Project Lookup Services should be properly configured and set up.

4. Usage Notes

4.1 Required steps to test the component

- Extract the component distribution.
- Follow Dependencies Configuration.
- Execute ‘ant test’ within the directory that the distribution was extracted to.

4.2 Required steps to use the component

The used web services from Client Project Lookup Services component should be configured properly. And the ClientProjectLookupServiceBridgeServlet should be deployed into a servlet container properly.

4.3 Demo

```
Assume the ClientProjectLookupServiceBridgeServlet is configured with
docs/web.xml in servlet container, and the deployed web application name
is “bridge”.
```

4.3.1 Get Client By Id

```
<html>
<head>

  <SCRIPT LANGUAGE="JavaScript">
  <!--
  // success callback function
  function onSuccess1(client) {
     // write the client name into the div with id="client1"
     var c = document.getElementById("client1");
     c.innerText = "client name: " + client.getName();
  }

  // error callback function
  function onError1(message) {
    // write the message into the div with id="client1"
    document.getElementById("client1").innerText = message;
  }

  // success callback function
  function onSuccess2(client) {
     // write the client into the div with id="client2"
     var c = document.getElementById("client2");
     c.innerText = "client name: " + client.getName();
  }

  // error callback function
  function onError2(message) {
    // write the message into the div with id="client2"
    document.getElementById("client2").innerText = message;
  }
  //-->
  </SCRIPT>
</head>

<body>
  <SCRIPT LANGUAGE="JavaScript">
  <!--
    // create a LookupService object
    var lookupService =
      new LookupService("http://localhost:8080/bridge/ajaxBridge");

    // get the client with id = 1
    lookupService.retrieveClient(1, onSuccess1, onError1);

    // get the client with id = 2
    lookupService.retrieveClient(2, onSuccess2, onError2);
  //-->
  </SCRIPT>

  <div id="client1">
  </div>
  <div id="client2">
  </div>
</body>
</html>
```

NOTE: The clients (with id = 1 / 2) will be retrieved asynchronously.

Assumes the client with id = 1 is retrieved successfully, and its name is “BBC”, then the div HTML element with id = “client1” will be updated to:

```
<div id=”client1”>
client name: BBC
</div>
```

Assumes the servlet fails to get the client with id = 2, and the error message is “Invalid client id.”, then the div HTML element with id = “client2” will be updated to:

```
<div id=”client2”>
Invalid client id.
</div>
```

4.3.2 Update Project Status

```
<html>
  <head>

  <SCRIPT LANGUAGE="JavaScript">
  <!--
  function onSuccess(status) {
     // write the new status name into the div with id="status"
     var s = document.getElementById("status");
     s.innerText = "status name: " + status.getName();
  }

  function onError(message) {
    // write the message into the div with id="status"
    document.getElementById("status").innerText = message;
  }
  //-->
  </SCRIPT>

  </head>

  <body>
  <SCRIPT LANGUAGE="JavaScript">
  <!--
    var service =
      new ProjectStatusService("http://localhost:8080/bridge/ajaxBridge");

    // create a ProjectStatus to update the name and description
    var status = new ProjectStatus();
    status.setId("1");
    status.setName("complete");
    status.setDescription("The complete status.");

    // update project status
    service.updateProjectStatus(status, onSuccess, onError);
  //-->
  </SCRIPT>

  <div id="status">
  </div>

  </body>
</html>
```

NOTE: The project status with id = 1 will be updated asynchronously.

If everything goes well, the project status data on server side will be updated with the new name and description, and the div HTML element with id = “status” will be updated to:

```
<div id=”status”>
status name: complete
</div>
```

The other JavaScript services are quite similar.

5 Future Enhancements

None.