# Client Project Management Javascript Bridge Component Specification

## 1. Design

This component provides an AJAX bridge, allowing JavaScript components or web-pages to interact with the Client Project Management 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 Management 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 Management Services component, and then converts returned values into an AJAX response which is returned to the JavaScript caller.

### 1.1 Design Patterns

We are using the ***delegate pattern*** since we are delegating the actual processing through our javascript services to the actual web services. We are also utilizing the ***strategy pattern*** since such actions as logging or configuration lookup and processing are done through pluggable interface definitions. The same applies to the services that we call since we plug them into the servlet.

### 1.2 Industry Standards

- JSON
- Ajax
- Java Servlet
- HTTP (for servlets and ajax)
- JavaBeans (which the Client Project Management uses for its entities)

### 1.3 Required Algorithms

#### *1.3.1 Logging*

The AjaxBridgeServlet should log all calls at DEBUG level, including the name of the service method which is to be called, and information to identify the objects involved - object ids but not the complete parameter data.
Also, all thrown exceptions should be logged with ERROR 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 Java Script Functions*

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

#### *1.3.3 Evaluate JSON string in Java Script*

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 JavaScript Service*

The JavaScript services (`ProjectService`, `ClientService` and `CompanyService`) will send AJAX requests to a configured servlet url, and the returned response will be JSON object in string representation.

#### 1.3.4.1 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 indicate the operation succeeds, and the "json" property represents the JSON object corresponding to the returned value of specific service method. The "json" property is not present if the corresponding service method doesn't return anything.

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

NOTE: The corresponding 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.

#### 1.3.4.2 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.

### *1.3.5 Handle the request in AjaxBridgeServlet*

The requests sent by the Java Script Services Classes will always have the "service" and "method" parameters.

The "service" parameter value can be "ProjectService", "ClientService" and "CompanyService", and they correspond to the `projectService`, `clientService`, and `companyService` field services respectively in the servlet.

The "method" parameter value represents the method name of the corresponding service. For example, if the "service" parameter value is "ProjectService", and the "method" parameter value is "createProject", then the projectService.createProject should be called to send the request to the service.

And depending on the service method to call, we should also extract corresponding method arguments' values from the request parameters. Take the `projectService.createProject` method as an example:

1. The request contains the "ProjectService" 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 projectService.createProject method with the Project object created above.
5. Get the returned value, and convert it into a JSONObject.
6. Call the jsonEncoder.encode to encode the JSONObject created in step5 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 in step7.

   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.

NOTE: For service method that doesn't return any value, we will write:
`{ "success" : true }` into the response if no exception occurs.

#### 1.3.5.1 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 Java Script Object in its `toJSON()` method, and the Java Script 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 is an example for ProjectStatus js entity:

Consider this received JSON string:

```
{ "id" : "1",
  "createUsername" : "ivern",
  "createDate" : "1163531522089",
  "modifyUsername" : "ivern",
  "modifyDate" : "1163531525089",
  "name" : "suspended",
  "deleted" : false",
  "description" : "suspended project status"
}
```

Dates are represented as milliseconds.

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

```
// Create a ProjectStatus entity:
ProjectStatus projectStatus = new ProjectStatus();
// Populate the project entity with data from the JSONObject.
projectStatus.setId(jsonProjectStatus.getLong("id"));
projectStatus.setCreateUsername(jsonProjectStatus.getString("createUsername"));
projectStatus.setCreateDate(jsonProjectStatus.getLong("createDate"));
projectStatus.setModifyUsername(jsonProjectStatus.getString("modifyUsername"));
projectStatus.setModiyDate(jsonProjectStatus.getLong("modifyDate"));
```

```
projectStatus.setName(jsonProjectStatus.getString(“name”));
projectStatus.setIsDeleted(jsonProjectStatus.getBoolean(“deleted”));
projectStatus.setDescription(jsonProjectStatus.getString(“description”));
// And the conversion from project to a JSONObject is similar
JSONObject jsonObj = new JSONObject();
jsonObj.setLong(“id”, projectStatus.getId());
jsonObj.setString(“createUsername”, projectStatus.getCreateUsername());
. . .
jsonObj.setString(“description”, projectStatus.getDescription());
```

1.3.5.2 Implement the doPost method

If “service” parameter value is “ProjectService”
  If “method” parameter value is “createProject”
    other request parameters:
    “project” - its value is a JSON string for a project, it should be converted to a Project object to pass into the service method.
    service method to call: projectService.createProject
    $json = the JSON string of the returned Project from the service method.
  Else if “method” parameter value is “updateProject”
    other request parameters:
    “project” - its value is a JSON string for a project, it should be converted to a Project object to pass into the service method.
    service method to call: projectService.updateProject
    $json = the JSON string of the returned Project from the service method.
  Else if “method” parameter value is “deleteProject”
    other request parameters:
    “project” - its value is a JSON string for a project, it should be converted to a Project object to pass into the service method.
    service method to call: projectService.deleteProject
    $json = the JSON string of the returned Project from the service method.
  Else if . . . // similar steps as above
  End if
Else if “service” parameter value is “ClientService”
  If “method” parameter value is “createClient”
    other request parameters:
    “client” - its value is a JSON string for a client, it should be converted to a Client object to pass into the service method.
    service method to call: clientService.createClient
    $json = the JSON string of the returned Client from the service method.
  Else if “method” parameter value is “updateClient”
    other request parameters:
    “client” - its value is a JSON string for a client, it should be converted to a Client object to pass into the service method.
    service method to call: clientService.updateClient
    $json = the JSON string of the returned Client from the service method.
  Else if “method” parameter value is “deleteClient”
    other request parameters:
    “client” - its value is a JSON string for a client, it should be converted to a Client object to pass into the service method.
    service method to call: clientService.deleteClient
    $json = the JSON string of the returned Client from the service method.
  Else if . . . // similar steps as above
  End if
Else if “service” parameter value is “CompanyService”
  // note that the rest of the code here is basically the same and follows the
  // same rules. Developers should have no issues here.
End if

NOTE: The developers can also consult the request parameters documented in the corresponding Java Script Service Classes.

All parameters are required except the one mentioned above. 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 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.

## 1.4 Component Class Overview

### *1.4.1 Package com.topcoder.clients.bridge.management (Java)*

- **ClientProjectManagementAjaxBridgeServlet**: This class extends the HttpServlet class, and it will decode AJAX requests from the JavaScript part into parameters used to call API methods in the Widget Services Wrapper component, which will make service calls. Returned values should be encoded into the AJAX response.

### *1.4.2 Package js.topcoder.clients.bridge.management (Java Script)*

- **ProjectService**: This JavaScript class defines the operations related to the project service.
- **ClientService**: This JavaScript class defines the operations related to the client service.
- **CompanyService**: This JavaScript class defines the operations related to the company service.

### *1.4.3 Package js.topcoder.clients.bridge.management.model (Java Script)*

- **AuditableEntity**: This is not really a javascript class but more like a definition of a base class which stipulates all the data that must be available to any entity defined in the model.
- **ProjectStatus**: This Java Script object represents the project status entity data.
- **ClientStatus**: This Java Script object represents the client status entity data.
- **Project**: This Java Script object represents the project entity data.
- **Client**: This Java Script object represents the client entity data.
- **Company**: This Java Script object represents the company entity data.

## 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 `AjaxBridgeServlet` 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 server 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 Java Script Entity classes are all mutable and not thread-safe.
- The Java Script 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 Java Script entity objects to different service methods especially when the entity objects will be changed externally by user, which might cause unexpected result. (NOTE: The Java Script entity objects passed into Java Script Service classes will not be changed.)

## 2. Environment Requirements

### 2.1 Environment

- Java 1.5+
- EJB 3 / JBoss 4.2
- Any Servlet Container
- Javascript

### 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.
- **Configuration API 1.0** - Used to configure the object factory.
- **Configuration Persistence 1.0** - Used to load configuration from persistence.
- **Object Factory 2.0.1** - Used to create web service client classes.
- **Object Factory Configuration API Plugin 1.0** - Used to create the Object Factory.
- **Client Project Management Services 1.0 –** Used for all the java web service definitions, client service definitions as well as the java bean entity definitions utilized in this bridge component.

### 2.3 Third Party Components

None.

## 3. Installation and Configuration

### 3.1 Package Name

*3.1.1 JavaScript*

js.topcoder.clients.bridge.management
js.topcoder.clients.bridge.management.model

*3.1.2 Java*

com.topcoder.clients.bridge.management

### 3.2 Configuration Parameters

For AjaxBridgeServlet (web.xml)

| 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 AjaxBridgeServlet. | 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. |
| projectServiceClientKey | The key used to create the ProjectServiceClient object from Object Factory. ***Optional***. Default to "projectServiceClient" | Must be non-empty string if present. |
| clientServiceClientKey | The key used to create the ClienterviceClient object from Object Factory. ***Optional***. Default to "clientServiceClient" | Must be non-empty string if present. |
| companyServiceClientKey | The key used to create the CompanyServiceClient object from Object Factory. ***Optional***. Default to "companyServiceClient" | 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

None.

## 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 services from *Client Project Management Services 1.0* component should be configured properly. And the AjaxBridgeServlet should be deployed into a servlet container properly.

## 4.3 Demo

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

### 4.3.1 *Update Project*

Assume that we have the following Project in persistence (only partial fields relevant to the demo are listed) We will update this project with some other data.

```
id: “1”
active: true
name: “London Ferris Wheel”
description: “The Ferris wheel experience”
```

Here is the actual demo part:

```
<html>
  <head>
    <SCRIPT LANGUAGE="JavaScript">
    <!--
     function onSuccess(project) {
         // write the project into the div with id="project"
         var proj = document.getElementById("project");
         proj.innerText = "project name: " + project.getName();
     }
     function onError(message) {
         // write the message into the div with id="project"
         document.getElementById("project").innerText = message;
     }
    //-->
    </SCRIPT>
  </head>
  <body>
  <SCRIPT LANGUAGE="JavaScript">
  <!--
     var projectService =
        new ProjectService("http://localhost:8080/bridge/ajaxBridge");

     // create a Project to update:
    var project = new Project();
    project.setId("1");
    project.setName("Millennium Wheel");
    project.setDescription("Creation of the Millennium Wheel Experience");
    project.setModifyUsername(“mess”);
    project.setModiyDate(new Date().milliseconds);
    project.setActive(false);


    // update project
    projectService.updateProject(project, onSuccess, onError);
  //-->
  </SCRIPT>
  <div id="project">
  </div>
  </body>
```

```
</html>
```

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

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

```
<div id=”project”>
project name: Millennium Wheel
</div>
```

The other Java Script services are quite similar and will not be repeated here.

The designer give us a demo to show it with html.

Before run this demo, please set servlet correctly and start Tomcat.

Here is the demo in JsUnit:

```
    /**
     * <p>Demo Test</code>.</p>
     */
    function testDemo() {
        var url = "http://localhost:8080/topcoder/servlet";
        var projectService = new
js.topcoder.clients.bridge.management.ProjectService(url);

        // create a Project to update:
        var project = new
js.topcoder.clients.bridge.management.model.Project();
        project.setID("1");
        project.setName("Millennium Wheel");
        project.setDescription("Creation of the Millennium Wheel
Experience");
        project.setModifyUsername("mess");
        project.setModifyDate(new Date());
        project.setIsActive(false);
        // update project
        projectService.updateProject(project, onSuccess, onError);
    }

    /**
     * The function use to test.
     */
    function onSuccess(/* Object */ obj) {
        alert("Demo is Success!");
    }

    /**
     * The function use to test.
     */
    function onError(/* String */ message) {
        alert("onError:" + message);
          }
```

## 5 Future Enhancements

None.