Widget Bridge 1.0 Component Specification

1. Design

This component provides an AJAX bridge, allowing JavaScript components or web-pages to interact with service components – Prerequisite Service, Project Service as well as Studio Service.

The component has two distinct parts. Firstly a JavaScript part which provides an API mirroring that of the service components. This API interacts with the other part, a Java servlet, through AJAX requests. The servlet translate the requests into parameters which are used to call into the service APIs, and then converts returned values into an AJAX response which is returned to the JavaScript part.

The Java Script service classes will send the AJAX requests, and process the received AJAX response, and then notify user through callback functions with parsed response data. The AjaxBridgeServlet class represents the servlet to handle the AJAX request and return the response in JSON format.

See: http://www.topcoder.com/wiki/display/docs/Java+Custom+Widget+Bridge

NOTE: this component development is based on widget webservice bridge 1.0 and it still keeps its flexible architecture and it uses service interfaces as delegates so that it can be easily plugged into EJB clients , which will be also extending service interfaces, to make calls to various services.

During this development, since all services are still having some issues , demo is still using mock services and as long as service interfaces are not changed, it can be easier to be used in EJB environment or other means. It has been confirmed with PMs through emails.

1.1 Design Patterns

None.

1.2 Industry Standards

AJAX, JSON

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.

And 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, PrerequisiteService and StudioService) 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 StudioService.uploadDocumentForContest

NOTE: It is not implemented yet and it is scheduled to be done in finalfix. See the forum.

As we cannot upload file to the server using the AJAX, so we need a different approach in the StudioService.uploadDocumentForContest method.

First we will generate the following iframe HTML fragment dynamically via JavaScript (using document.createXXX functions):

```
<iframe style="display:none" src="about:blank" onload="loadFrame()">
 <form method="POST" action="$servletUrl">
   <input type="hidden" name="service" value="studio" />
   <input type="hidden" name="method" value="uploadDocumentForContest" />

   <input type="hidden" name="documentID" value="$documentID" />
   <input type="hidden" name="contestID" value="$contestID" />
   <input type="hidden" name="description" value="$description" />
   <input type="file" name="document" value="$fileName" />
 </form>
</iframe>
```

Where the $servletUrl is the requested servlet url, and the $documentID, $contestID, $description and $fileName are corresponding property values of the passed-in UploadedDocument JavaScript object. The value of $fileName means the local file name to upload.

The loadFrame JavaScript function should be like:

```
function loadFrame() {
  If the iframe's src attribute value is not "about:blank" any more
    It means the form inside the iframe is submitted,
    as the src attribute value is changed.

    Get the iframe's inner string content, and then evaluate it into a
    JSON object (assume its variable name is jsonResp).
    The returned JSON object will has the format mentioned in 1.3.2.

    If jsonResp.success is false
        Call onError callback function with jsonResp.error
    Else
        Create an UploadedDocument JavaScript object with jsonResp.json.
        Call onSuccess with the created UploadedDocument object.
    EndIf
  EndIf
}

NOTE: loadFrame should be written as an inner JavaScript function of
uploadDocumentForContest, so it can access uploadDocumentForContest's
arguments.
```

1.3.6 *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 "project", "prerequisite" and "studio", and they correspond to the projectService, prerequisiteService, and studioService services respectively.

The "method" parameter value represents the method name of the corresponding service. For example, if the "service" parameter value is "project", 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 "project" 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 step6 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.

1.3.6.1 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 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 gives an example:

Received JSON string:
```
{"projectID" : 1, "name" : "topcoder", "description" : "tc design" }
```

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

```
// Create a Project entity:
Project project = new Project();

// Populate the project entity with data from the JSONObject.
project.setProjectID(jsonProject.getLong(“projectID”));
project.setName(jsonProject.getString(“name”));
project.setDescription(jsonProject.getString(“description”));

// And the conversion from project to a JSONObject is similar
JSONObject jsonObj = new JSONObject();
jsonObj.setLong(“projectID”, project.getProjectID());
jsonObj.setString(“name”, project.getName());
jsonObj.setString(“description”, project.getDescription());
```

1.3.6.2 Implement the doPost method

If “service” parameter value is “project”
  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:
    “projectID” - its value is projectID string value, it should be converted to a long value to pass into the service method.
    service method to call: projectService.deleteProject
    the service method has no return value.

Else if “method” parameter value is “getProject”
other request parameters:
“projectID” - its value is projectID string value, it should be converted to a long value to pass into the service method.
service method to call: projectService.getProject
$json = the JSON string of the returned Project from the service method.
Else if “method” parameter value is “getProjectsForUser”
other request parameters:
“userID” - its value is userID string value, it should be converted to a long value to pass into the service method.
service method to call: projectService.getProjectsForUser
$json = the JSON string of the returned projects from the service method.
(A JSONArray should be created, and then encoded to JSON string).

Else if “method” parameter value is “getAllProjects”
NO other request parameters:
service method to call: projectService.getAllProjects
$json = the JSON string of the returned projects from the service method.
(A JSONArray should be created, and then encoded to JSON string).
End if
Else if “service” parameter value is “prerequisite”
If “method” parameter value is “getAllPrerequisiteDocuments”
other request parameters:
“clientID” - its value is clientID string value, it should be converted to a long value to pass into the service method.
service method to call: prerequisiteService. getAllPrerequisiteDocuments
$json = the JSON string of the returned documents from the service method.
(A JSONArray should be created, and then encoded to JSON string).
Else if “method” parameter value is “getPrerequisiteDocuments”
other request parameters:
“competitionID” -competition id
“roleID” – role id
service method to call: prerequisiteService. getPrerequisiteDocuments
$json = the JSON string of the returned documents from the service method.
(A JSONArray should be created, and then encoded to JSON string).
Else if “method” parameter value is “getPrerequisiteDocument”
other request parameters:
“documentID” – document long id
“version” -- version, it is integer value
service method to call: prerequisiteService. getPrerequisiteDocument
$json = the JSON string of returned document
Else if “method” parameter value is “recordMemberAnswer”
Other parameters:
“competitionID” : competition long id
“timestamp” : timestamp
“agrees” : Boolean value
“roleID” : role id
“prerequisteDocument”: JSON string of PrerequisiteDocument
service method to call: prerequisiteService.recordMemberAnswer
End if
Else if “service” parameter value is “studio”
If “method” parameter value is “createContest”
other request parameters:
“contest” - its value is a JSON string for a contest, it should be converted to a Contest object to pass into the service method.

"projectID" - its value is projectID string value, it should be converted to a long value to pass into the service method.
service method to call: studioService.createContest
the service method has no return value.

Else if "method" parameter value is "getContest"
other request parameters:
"contestID" - its value is contestID string value, it should be converted to a long value to pass into the service method.
service method to call: studioService.getContest
$json = the JSON string of the returned Contest from the service method.

Else if "method" parameter value is "getContestsForClient"
other request parameters:
"clientID" - its value is clientID string value, it should be converted to a long value to pass into the service method.
"contestStatusID" - its value is contestStatusID string value, it should be converted to a long value to pass into the service method.
service method to call: studioService.getContestsForClient
$json = the JSON string of the returned Contests from the service method.
(A JSONArray should be created, and then encoded to JSON string).

Else if "method" parameter value is "getContestsForProject"
other request parameters:
"projectID" - its value is projectID string value, it should be converted to a long value to pass into the service method.
service method to call: studioService.getContestsForProject
$json = the JSON string of the returned Contests from the service method.
(A JSONArray should be created, and then encoded to JSON string).

Else if "method" parameter value is "updateContestStatus"
other request parameters:
"contestID" - its value is contestID string value, it should be converted to a long value to pass into the service method.
"newStatusID" - its value is newStatusID string value, it should be converted to a long value to pass into the service method.
service method to call: studioService.updateContestStatus
$json = the JSON string of the returned Contest from the service method.

Else if "method" parameter value is "uploadDocumentForContest"
other request parameters:
"documentID" - its value is documentID string value, it should be converted to a long value.
"contestID" - its value is contestID string value, it should be converted to a long value.
"description" - its value is description string value.

call fileUpload = new MemoryFileUpload(fileUploadConfig).
then call input = fileUpload.uploadFiles(request).getAllUploadedFiles()[0].getInputStream() to get the input stream of the uploaded file
read data from input into a byte[] array, and close input stream.
create a UploadedDocument object with the values retrieved above, this object should be passed to the service method.
service method to call: studioService.uploadDocumentForContest
$json = the JSON string of the returned UploadedDocument from the web service method
(NOTE: only the documentID, contestID, and description values are set to the JSON string according to the corresponding JavaScript Object.)

Else if "method" parameter value is "removeDocumentFromContest"
other request parameters:

"document" - its value is a JSON string, it should be converted to a
UploadedDocument object to pass into the service method.
(NOTE: the file attribute of UploadedDocument doesn't need to be set.)
service method to call: studioService.removeDocumentFromContest
the service method has no return value.

Else if "method" parameter value is "retrieveSubmissionsForContest"
other request parameters:
"contestID" - its value is contestID string value, it should be converted to a long
value to pass into the service method.
service method to call: studioService.retrieveSubmissionsForContest
$json = the JSON string of the returned Submissions from the service method.
(A JSONArray should be created, and then encoded to JSON string).

Else if "method" parameter value is "updateSubmission"
other request parameters:
"submission" - its value is a JSON string, it should be converted to a
Submission object to pass into the service method.
service method to call: studioService.updateSubmission
the service method has no return value.

Else if "method" parameter value is "updateContest"
other request parameters:
"contest" - its value is a JSON string, it should be converted to a
Contest object to pass into the service method.
service method to call: studioService.updateContest
the service method has no return value.

Else if "method" parameter value is "retrieveSubmission"
other request parameters:
"submissionID" - its value is submissionID string value, it should be converted to a
long value to pass into the service method.
service method to call: studioService.retrieveSubmission
$json = the JSON string of the returned Submission from the service method.

Else if "method" parameter value is "removeSubmission"
other request parameters:
"submissionID" - its value is submissionID string value, it should be converted to a
long value to pass into the service method.
service method to call: studioService.removeSubmission
the service method has no return value.

Else if "method" parameter value is "retrieveAllSubmissionsByMember"
other request parameters:
"userID" - its value is userID string value, it should be converted to a long
value to pass into the service method.
service method to call: studioService.retrieveAllSubmissionsByMember
$json = the JSON string of the returned Submissions from the service method.
(A JSONArray should be created, and then encoded to JSON string).

Else if "method" parameter value is "getContestCategories"
other request parameters:
"parameters" - its value is JSON string of an array of string values. (e.g. ["v1", "v2"])
it should be decoded into JSONArray with string values, and then
extract string values from it. **Optional Parameter.**
service method to call: studioService.getContestCategories
$json = the JSON string of the returned ContestCategories from the service
method.
(A JSONArray should be created, and then encoded to JSON string).

Else if "method" parameter value is "getSubmissionFileTypes"
NO other request parameters
service method to call: studioService.getSubmissionFileTypes

$json = the JSON string of the returned string values from the service method.
(A JSONArray should be created, and then encoded to JSON string).
Else if “method” parameter value is “getContestStatuses”
NO other request parameters
service method to call: studioService.getContestStatuses
$json = the JSON string of the returned ContestStatuses from the service method.
(A JSONArray should be created, and then encoded to JSON string).
End If
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.widgets.bridge (Java)

- **AjaxBridgeServlet**: 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.widgest.bridge (Java Script)

- **ProjectService**: This JavaScript class defines the operations related to the project service.
- **PrerequisiteService**: This JavaScript class defines the operations related to the Prerequisite service.
- **StudioService**: This JavaScript class defines the operations related to the studio service.
- **Project**: This Java Script object represents the project data.
- **Submission**: This Java Script object represents the submission data.
- **Prize**: This Java Script object represents the Prize data.
- **UploadedDocument**: This Java Script object represents the UploadedDocument data.
- **Contest**: This Java Script object represents the Contest data.
- **ContestCategory**: This Java Script object represents the Contest Category data.
- **ContestStatus**: This Java Script object represents the Contest Status data.
- **ContestPayload**: This Java Script object represents the Contest Payload data.
- **PrerequsiteDocument**: This Java Script object represents PrerequisiteDocument 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

2.2 TopCoder Software Components

- **Logging Wrapper 2.0** - Used to log invocation information and exceptions.
- **AJAX Processor 1.0** - Used to send AJAX request and receive response.
- **JSON Object 1.0** - Used to convert data between the JSON string and JSON java objects.
- **File Upload 2.1** - Used to read uploaded file content.
- **Configuration API 1.0** - Used to create file upload.
- **Configuration Persistence 1.0** - Used to load configuration from persistence.
- **Object Factory 2.0.1** - Used to create service wrapper classes.
- **Object Factory Configuration API Plugin 1.0** - Used to create the Object Factory.
- **Studio Service 1.0**
- **Prerequisite Service 1.0**

- **Project Service 1.0**

2.3 Third Party Components

None.

3. Installation and Configuration

3.1 Package Name

js.topcoder.widget.bridge (Java Script)

com.topcoder.widget.bridge (Java)

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.
projectServiceKey	The key used to create the ProjectService object from Object Factory. Optional. Default to “projectService”	Must be non-empty string if present.
prerequisiteServiceKey	The key used to create the PrerequisitetService object from Object Factory. Optional. Default to “prerequisiteService”.	Must be non-empty string if present.
studioServiceKey	The key used to create the StudioService object from Object Factory. Optional. Default to “studioService”.	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.
fileUploadNamespace	Represents the namespace used to get the ConfigurationObject from ConfigurationFileManager to create the MemoryFileUpload 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 Widget Webservices Wrapper 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 "bridge".
```

4.3.1 Get Project

```
<html>
<head>

  <SCRIPT LANGUAGE="JavaScript">
  <!--
  // success callback function
  function onSuccess1(project) {
     // write the project into the div with id="project1"
     var proj = document.getElementById("project1");
     proj.innerText = "project name: " + project.getName();
  }

  // error callback function
  function onError1(message) {
    // write the message into the div with id="project1"
    document.getElementById("project1").innerText = message;
  }

  // success callback function
  function onSuccess2(project) {
     // write the project into the div with id="project2"
     var proj = document.getElementById("project2");
     proj.innerText = "project name: " + project.getName();
  }

  // error callback function
  function onError2(message) {
    // write the message into the div with id="project2"
    document.getElementById("project2").innerText = message;
  }
```

```
  //-->
  </SCRIPT>
</head>

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

    // get the project with projectID = 1
    projectService.getProject(1, onSuccess1, onError1);

    // get the project with projectID = 2
    projectService.getProject(2, onSuccess2, onError2);
  //-->
  </SCRIPT>

  <div id="project1">
  </div>
  <div id="project2">
  </div>
</body>
</html>
```

NOTE: The projects (with projectID = 1 / 2) will be retrieved asynchronously.

Assumes the project with projectID = 1 is retrieved successfully, and its projectName is “topcoder”, then the div HTML element with id = “project1” will be updated to:

```
<div id=”project1”>
project name: topcoder
</div>
```

Assumes the servlet fails to get the project with projectID = 2, and the error message is “Project Service Not Available”, then the div HTML element with id = “project2” will be updated to:

```
<div id=”project2”>
Project Service Not Available
</div>
```

4.3.2 *Update Project*

```
<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 the name and description
    var project = new Project();
    project.setProjectID("1");
    project.setName("newName");
    project.setDescription("newDesc");

    // 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 name and description, and the div HTML element with id = “project” will be updated to:

```
<div id=”project”>
project name: newName
</div>
```

4.3.3 *Upload document for contest*

```
<html>
  <head>
```

```
  <SCRIPT LANGUAGE="JavaScript">
  <!--
  function onSuccess(doc) {
     // write the doc's description into the div with id="doc"
     var d = document.getElementById("doc");
     d.innerText = "document desc: " + doc.getDescription();
  }

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

  </head>

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

    // create a UploadedDocument
    var doc = new UploadedDocument();
    doc.setDocumentID(1);
    doc.setContestID(1);
    doc.setFileName("c:\test.txt");
    doc.setDescription("TEST");

    // upload document
    studioService.uploadDocumentForContest(doc, onSuccess, onError);
  //-->
  </SCRIPT>

  <div id="doc">
  </div>

  </body>
</html>
```

NOTE: The `UploadedDocument` object containing the local file: "c:\test.txt" will be uploaded. And if everything goes well, the document will be added to the contest (with contestID = 1) and the div HTML element with id = "doc" will be updated to:

```
<div id="doc">
document desc: TEST
</div>
```

The other Java Script services are quite similar.

5 Future Enhancements

None.