Online Review Ajax Support 1.0 Component Specification

1. Design

The Online Review application uses AJAX features to enhance user experience. The component provides the server side support for implementing the AJAX interactions. AJAX will be used to set scorecard status, load timeline template, set timeline notification, place and resolve appeals. The AJAX service will be provided as Java servlets.

Approach

This component was designed with the future extensibility in mind; it defines two main parts in order to fulfill a requested Ajax operation.

- The main HTTP servlet – which is implemented by the class *AjaxSupportServlet*, and is responsible for parsing an Ajax request, forwarding it to the right Ajax request handler, and then sending back the Ajax response to the client.
- Ajax request handlers – each one of this handler implements the *AjaxRequestHandler* interface, and process a single operation type.
  Upon receipt of an Ajax request the handler service that request and return an Ajax response containing the result of the operation.

In order to be generic, the Ajax request consists of:

- Type – the type of the Ajax request, which is used by the *AjaxSupportServlet* class to find the correct handler in order to service the request.
- Parameters – zero or more request parameters; each parameter consists of:
  - A unique name which is a not null and not empty *String* object.
  - A value which is a *String* object, or an empty *String*.

For instance the XML representation of the *"Set Timeline Notification"* operation request could be like this:

```
<?xml version="1.0" ?>
<request type="SetTimelineNotification">
      <parameters>
            <!-- the project ID parameter -->
            <parameter name="ProjectId">51</parameter>

            <!-- the notification status parameter put to "On" -->
            <parameter name="Status">On</parameter>
      </parameters>
</request>
```

Also, in order to be generic, the Ajax response consists of:

- Type – the type of the Ajax response which is the same as the Ajax request type.
- Status – the status of the operation which could be *"Success"* for successful operation or any other status like error statuses.

- Data – which could represent a result data, or an error message; this field is optional.

For instance the XML representation of the *"Resolve Appeal"* operation successful response could be like this:

```
<?xml version="1.0" ?>
<response type="ResolveAppeal">
      <!-- the status is "Success" and the data is the review score -->
      <result status="Success">0.982144</result>
</response>
```

This approach of using a generic Ajax request and response makes the interaction between the Ajax client and this component looks like an RPC (Remote Procedure Call) based on SOAP messages.

Also, the code of Ajax client side will be generic; for instance a small script library could be written to create an Ajax request and to get data from an Ajax response.

This component defines five Ajax request handlers implementing these operations:

- Set scorecard status
- Load timeline template
- Set timeline notification
- Place appeal
- Resolve appeal

The details of input parameters of these operations are discussed under the configuration parameters section.

New handlers could be added in order to fulfil other operation types.

The *AjaxSupportServlet* class implements only the *"doPost"* method, this makes sense since the Ajax client must send an XML document to be processed. A good side effect of this is that there is no need to include cache control HTTP header in the HTTP response since the browser won't cache the returned data.

The configuration approach was to use both *Configuration Manager* and *Object Factory* components, see the configuration parameters section for the details.

The *Logging Wrapper* component was used in order to log any successful or failed operation.

1.1 Design Patterns

Strategy – used to permit the Ajax request handlers to be changed, updated, or even extended by creating new ones without affecting the logic of the main servlet.

1.2 Industry Standards

- XML
- AJAX
- JAXP
- Servlet 2.4

1.3 Required Algorithms

1.3.1 Getting a schemas files

In order to create the *File* object for the XSD schema, this method can be used:

```
File getFile(String filePath) {
        // get the URL of the resource
        URL url = SAXTest.class.getClassLoader().getResource(filePath);

        // return the file
        return new File(url.getFile());
}
```

Example of usage:

```
// get the schema file used to validate incoming Ajax requests
File xsdFile=getFile("com/cronos/onlinereview/ajax/AjaxRequestSchema.xsd
");
```

1.3.2 Parsing an Ajax request

Developers could use either DOM or SAX to parse the Ajax request, but using SAX is the best solution because of its speed and small memory consumption.

```
// JAXP properties
String JAXP_SCHEMA_LANGUAGE =
            "http://java.sun.com/xml/jaxp/properties/schemaLanguage";
String JAXP_SCHEMA_SOURCE =
            "http://java.sun.com/xml/jaxp/properties/schemaSource";

// XSD schema language property values
String W3C_XML_SCHEMA = "http://www.w3.org/2001/XMLSchema";

// creating a SAX factory
SAXParserFactory factory = SAXParserFactory.newInstance();
factory.setValidating(true);
factory.setNamespaceAware(true);

// defining the SAX handler
AjaxRequestParser handler = new AjaxRequestParser();

try {
```

```
        // creating a parser
        SAXParser parser = factory.newSAXParser();

        // setting the schema language to XSD
        parser.setProperty(JAXP_SCHEMA_LANGUAGE, W3C_XML_SCHEMA);

        // setting the schema source
        parser.setProperty(JAXP_SCHEMA_SOURCE, getFile(
          "com/cronos/onlinereview/ajax/AjaxRequestSchema.xsd"));

        // creating the source from the XML message
        InputSource source= new InputSource(new StringReader(message));

        // parsing and validating the request XML
        parser.parse(source, handler);

        // getting the Ajax request from the handler
        AjaxRequest request=handler.getRequest();
}
// catch exceptions
catch(...){}

// use the handler to get the results if the XML message is valid
handler. (....)
```

The *AjaxRequestParser* class is defined like this:

```
// this is the Ajax request parser class
static class AjaxRequestParser extends DefaultHandler{

        // the request type attribute
        private String requestType;

        // the request parameters
        private final Map requestParameters=new HashMap();

        // the request parameter name
        private String parameterName;

        // buffer used to collect the parameter value
        private StringBuffer sb;

        // the parsed AjaxRequest object
        private AjaxRequest request;

        // handle the start of an XML element
        public void startElement(String uri, String localName,
        String qName, Attributes attributes) throws SAXException {
            if(qName.equals("request")){
                if(attributes!=null){
                    requestType=attributes.getValue("type");
                }
            }
            else if(qName.equals("parameter")){
```

```
                if(attributes!=null){
                    parameterName=attributes.getValue("name");
                }
            }
            else{
                sb=null;
            }
        }

        // handle the content of an XML element
        public void characters(char[] ch, int start, int length)
        throws SAXException {
            if(sb==null){
                sb=new StringBuffer();
            }
            sb.append(ch,start,length);
        }

        // handle the end of an XML element
        public void endElement(String uri, String localName,
        String qName) throws SAXException {
            if(qName.equals("request")){
                request=new AjaxRequest(requestType,requestParameters);
            }
            else if(qName.equals("parameter")){
                String parameterValue=sb.toString();

                requestParameters.put(parameterName,parameterValue);
            }
        }

        // throw an exception for any XML validation error
        public void error(SAXParseException e) throws SAXException {
            throw e;
        }

        // returns the parsed AjaxRequest object
        public AjaxRequest getRequest() {
            return request;
        }
}
```

1.3.3 *Parsing and Formatting a Date*

```
// creating a simple date formatter
SimpleDateFormat formatter=new SimpleDateFormat("MM.dd.yyyy h:mm a");

// to parse a date from a string
Date date=formatter.parse(dateString);


// to output a date as a string when generating XML
formatter.format(date);
```

1.3.4 *Generating XML strings*

In order to create XML files, a simple string concatenation algorithm will be sufficient.

```
// first create a StringBuffer
StringBuffer sb=new StringBuffer();

// append the XML header if required so
sb.append("<?xml version=\"1.0\" ?>");

// append the XML elements to buffer
............

// transform the buffer to a String
String xml=sb.toString();
```

1.3.5 *Using Object Factory*

```
// create config manager specification factory
SpecificationFactory specFactory = new
ConfigManagerSpecificationFactory("com.cronos.onlinereview.ajax.factory"
);

// create an object factory that uses only the specification
ObjectFactory factory=new
ObjectFactory(specFactory,ObjectFactory.SPECIFICATION_ONLY);
```

The object factory must be used with *String* keys in order to create Ajax request handlers.

```
// create the AjaxRequestHandler object using a String key, here a
// ResolveAppeal
AjaxRequestHandler handler =
(AjaxRequestHandler)factory.createObject("ResolveAppeal");
```

The object factory must be used with *Class* keys in order to create manager classes.

```
// For resource manager class
ResourceManager manager = (AjaxRequestHandler)
factory.createObject(ResourceManager.class);
```

1.3.6 *XSD files*

All the XSD files are within this component JAR file, they are documented; please use the appropriate tool to read the documentation.

1.3.7 *Request/Response sample XML*

To see sample XML for all the operation handled by this component please refer to the Demo section and to the XSD files.

1.4 Component Class Overview

*1.4.1 **Package** com.cronos.onlinereview.ajax*

AjaxSupportServlet
Main servlet class of the component; this class extends *HttpServlet* class in order to process Ajax requests and produce Ajax responses.

This class keeps a map of all Ajax request handlers, and when the *"doPost"* method is called it follows these steps to process the request:

1. Get the user ID from the *HttpSession*.
2. Parse the Ajax request XML stream to produce an *AjaxRequest* object.
3. Pass the user id and the Ajax request to the correct request handler.
4. Write back the Ajax response XML produced by the request handler.

When the Ajax request is incorrect, invalid or the target handler was not found, then this servlet will return an error response to the client, and log that error using the *Logging Wrapper* component.

AjaxRequestHandler <<interface>>
Defines the contract an Ajax request handler must implement in order to process Ajax requests. This interface defines only one method which is the *"service"* method.

An Ajax request handler's *"service"* method will take an *AjaxRequest*, and a user ID as parameter in order to process the request from that user and produce the corresponding *AjaxResponse* object.

A successful servicing operation must return an *AjaxResponse* object with the *"Success"* status, and optionally the result data.

Any servicing error must be returned as an *AjaxResponse* object with an error specific status, and optionally the error message or data.

Any successful or failed operation is must be logged by the implementation classes using the *Logging Wrapper* component.

AjaxRequest
Represents an Ajax request sent by a client to the server; this request consists of:

- Type – the type of the Ajax request, which is used by the *AjaxSupportServlet* class to find the correct handler in order to service the request.
- Parameters – zero or more request parameters; each parameter consists of:
    - A unique name which is a not null and not empty *String* object.
    - A value which is a *String* object, or an empty *String*.

This class defines some helper methods to get a parameter value as a *Date* object, or a *long* data type.

Parsing an Ajax request is implemented by this class.

AjaxResponse
Represents an Ajax response produced by an Ajax request handler in order to be sent back to the requesting client; this response consists of:

- Type – the type of the Ajax response which is the same as the Ajax request type.
- Status – the status of the operation which could be *"Success"* for successful operation or any other status like error statuses.
- Data – which could represent a result data, or an error message; this field is optional.

Transforming an Ajax response to an XML document is implemented by this class.

1.4.2 ***Package*** *com.cronos.onlinereview.ajax.handlers*

CommonHandler <<abstract>>
Defines a common Ajax request handler capable of getting a user's role using its ID, and the Resource Management component; this class implements the *AjaxRequestHandler* interface, and keeps an instance of *RessourceManager* class in order to get resource related data.

This class main purpose is to simplify Ajax request handlers' implementation.

SetScorecardStatusHandler
Ajax request handler class which handles the *"Set Scorecard Status"* Ajax operation; this class extends the *CommonHandler* abstract class, and uses the resource manager defined in its parent class, plus an instance of the *ScorecardManager* class in order to implement the *"Set Scorecard Status"* operation.

The "*Set Scorecard Status"* operation involves:

1. Checking that the user has the global manager role.
2. Getting the scorecard using the *ScorecardManager* class.
3. Setting the scorecard status to *"Active"* or *"Inactive"* status.
4. Updating the scorecard using the *ScorecardManager* class.
5. Returning an Ajax response with "success" status.

The scorecard must be not in use when the scorecard status to be set is *"Inactive"*, otherwise an Ajax response will be returned with an error status.

Any successful or failed operation is logged using the *Logging Wrapper* component.

LoadTimelineTemplateHandler
Ajax request handler class which handles the *"Load Timeline Template"* Ajax operation; this class extends the *CommonHandler* abstract class, and uses the resource manager defined in its parent class, plus an instance of the *PhaseTemplate* class in order to implement the *"Load Timeline Template"* operation.

The *"Load Timeline Template"* operation involves:

1. Checking that the user has the global manager role.
2. Checking that the template exists.
3. Applying the template using the *PhaseTemplate* class.
4. Extracting and transforming the timeline data into an XML representation.
5. Returning an Ajax response with *"Success"* status and the timeline data's XML.

Any successful or failed operation is logged using the *Logging Wrapper* component.

SetTimelineNotificationHandler
Ajax request handler class which handles the *"Set Timeline Notification"* Ajax operation; this class extends the *CommonHandler* abstract class, and uses the resource manager defined in its parent class, plus an instance of the *ProjectManager* class in order to implement the *"Set Timeline Notification"* operation.

The "*Set Timeline Notification"* operation involves:

1. Cheking that one of these three conditions is satisfied:
   - The project has an extended property named *"Public"* having the value *"Yes"*. The project object is retrieved using the *ProjectManager* class.
   - The user has the global manager role.
   - The project is one of the user's projects; this is done using the *ProjectManager* class.
2. Adding/removing the timeline notification when the status parameter is *"On"*/*"Off"*.
3. Returning an Ajax response with *"Success"* status.

4. Any successful or failed operation is logged using the *Logging Wrapper* component.

ReviewCommonHandler <<abstract>>
Defines a common parent class to all handlers which service scorecard review related operations; this class extends *CommonHandler* class, and keeps instances of *ReviewManager*, *PhaseManager* classes.

This class main purpose is to simplify implementation of Ajax request handlers related to review scorecard operations.

PlaceAppealHandler
Ajax request handler class which handles the *"Place Appeal"* Ajax operation; this class extends the *ReviewCommonHandler* abstract class, and uses the managers defined in its parents classes, plus an instance of the *UploadManager* class in order to implement the *"Place Appeal"* operation.

The *"Place Appeal"* operation involves:

1. Checking that the review concerns this user; this involves:
    a. Getting the review using the *ReviewManager* class.
    b. Getting the submission using the *UploadManager* class.
    c. Comparing the user ID to the submitter ID.
2. Checking that the user has the *"Submitter"* role.
3. Checking that the author of the review has the "Reviewer" role.
4. Getting the *"Review"* phase for the project using the *PhaseManager* class.
5. Checking that the author of the review is assigned to the "Review" phase.
6. Getting the *"Appeals"* phase for the project using the *PhaseManager* class.
7. Checking that the *"Appeals"* phase has the status *"Open"*.
8. Adding the appeal comment to the review item; this comment has the *"Appeal"* comment type.
9. Updating the review using the *ReviewManager* class.
10. Returning an Ajax response with *"Success"* status.

Any successful or failed operation is logged using the *Logging Wrapper* component.

ResolveAppealHandler
Ajax request handler class which handles the *"Resolve Appeal"* Ajax operation; this class extends the *ReviewCommonHandler* abstract class, and uses the managers defined in its parents classes in order to implement the *"Resolve Appeal"* operation.

The *"Appeal Response"* operation involves:
Checking that the review concerns this user; this involves:

1. Getting the review using the *ReviewManager* class.
    a. Comparing the user ID to the author ID.
    b. Checking that the user has the *"Reviewer"* role.
2. Getting the *"Review"* phase for the project using the *PhaseManager* class.
3. Checking that the user is assigned to the *"Review"* phase.
4. Getting the *"Appeals Response"* phase for the project using the *PhaseManager* class.
5. Checking that the *"Appeals Response"* phase has the status *"Open"*.
6. Adding/updating the appeal response comment.
7. Updating the item response.
8. Update the appeal comment extra info with either *"Succeeded"* or *"Failed"* statuses.
9. Calculate the score of the review using the Review Score Calculator component.

10. Update the review using the *ReviewManager* class.
11. Returning an Ajax response with *"Success"* status and the review score.

Any successful or failed operation is logged using the *Logging Wrapper* component.

1.5 Component Exception Definitions

ConfigurationException
Represents an exception detected during configuration. This class extends the *BaseException* class.

This exception is:

- Thrown by all Ajax request handlers' constructors, if they fail to configure their internal state.
- Thrown by *AjaxSupportServlet* class when it fails to load configuration data, and to create Ajax request handlers; *AjaxSupportServlet*'s *"init"* method wraps this exception in a *ServletException*, and then thrown it.

RequestParsingException
Represents an exception detected during Ajax request parsing; the originator error could be due to:

- The XML document of the request is malformed.
- The XML document of the request is well formed, but it is invalid against the schema.

This exception is thrown by *AjaxRequest*'s "parse" method.

RoleResolutionException
Represents an exception used to wrap any checked exception thrown by the *ResourceManager* class.

This exception is thrown by many methods in the *CommonHandler* class.

1.6 Thread Safety

Thread safety for this component is required; this issue was treated like this:

- Any Ajax request handler implementation is required to be thread-safe.
- Default Ajax request handlers are thread safe since they are immutable classes. Manager classes defined in other components like *ResourceManager* class are thread safe.
- The *AjaxSupportServlet* class is thread-safe; all accesses to its internal state are read only ones.

- For *AjaxRequest* and *AjaxResponse* classes, they are both thread-safe since they are immutable classes.

2. Environment Requirements

2.1 Environment

- Development language: Java 1.4
- Compile target: Java 1.4

2.2 TopCoder Software Components

Configuration Manager 2.1.4 – used to simplify the process of getting the configuration parameters.

Object Factory 2.0 – used to create objects like manager classes, and Ajax request handlers in an abstract manner, thus decoupling the real implementation from its interface.

Base Exception 1.0 – used to provide a base exception class for all exception classes defined in this component.

Logging Wrapper 1.2 – used to provide logging capabilities to the main servlet, and to all Ajax request handlers classes.

Deliverable Management 1.0 – used to access uploads and submissions data.

Phase Management 1.0 – used to access project phases data.

Project Management 1.0 – used to access projects data.

Project Phase Template 1.0 – used to generate timeline data.

Resource Management 1.0 – used to access resource data.

Review Management 1.0 – used to access review data.

Review Data Structure 1.0 – defines the data structure of a review.

Scorecard Management 1.0 – used to access scorecard data.

Scorecard Data Structure 1.0 – defines the data structure of scorecard.

Review Score Calculator 1.0 – used to calculate the score of a committed review.

Project Phases 2.0 – defines the Phase and Project classes needed by this component.

NOTE: The default location for TopCoder Software component jars is../lib/tcs/COMPONENT_NAME/COMPONENT_VERSION relative to the component installation. Setting the tcs_libdir property in topcoder_global.properties will overwrite this default location.

2.3 Third Party Components

None.

NOTE: The default location for 3rd party packages is ../lib relative to this component installation. Setting the ext_libdir property in topcoder_global.properties will overwrite this default location.

3. Installation and Configuration

3.1 Package Name

- com.cronos.onlinereview.ajax
- com.cronos.onlinereview.ajax.handlers

3.2 Configuration Parameters

This component uses both *Configuration Manager*, and *Object Factory* components.

To support bout configuration techniques, this component defines two configuration namespaces, namely:

- com.cronos.onlinereview.ajax – used to define basic component configuration data.
- com.cronos.onlinereview.ajax.factory – used to define Object Factory configuration data.

3.2.1 Configuration under the namespace com.cronos.onlinereview.ajax

Parameter	Description	Values
UserIdAttributeName	Defines the attribute name used to retrieve the user ID from the HTTP session.	A not null and not empty string. No default value is defined.
Handlers	Defines a list of zero or more Ajax request handlers' names. Each name must correspond to the request type handled by the handler. Each name defined here must have its concrete class definition under the namespace ***"com.cronos.onlinereview.ajax.handlers".***	Zero or more not null and not empty strings. No default value is defined.

This listing demonstrates the required configuration under this namespace:

```
<Config name="com.cronos.onlinereview.ajax">
      <!-- defining the user id attribute name -->
      <Property name="UserIdAttributeName">
            <Value>UserId</Value>
      </Property>

      <!-- defining the Ajax request handlers names -->
      <Property name="Handlers">
            <Value>SetScorecardStatus</Value>
            <Value>LoadTimelineTemplate</Value>
            <Value>SetTimelineNotification</Value>
            <Value>PlaceAppeal</Value>
            <Value>ResolveAppeal</Value>
      </Property>
</Config>
```

3.2.2 *Configuration under the namespace com.cronos.onlinereview.ajax.factory*

This configuration concerns both the Ajax request handlers, and manager classes' concrete implementation definition.

- For Ajax request handlers – the key used to define the concrete implementation is a String defined under the parameter
- For manager classes – the key is the fully qualified name of the manager interface, but the dots are replaced with slashes.
  For instance if the class name was:
  "com.topcoder.management.resource.ResourceManager" then the key to use would be: *"com/topcoder/management/resource/ResourceManager"*.

This listing demonstrates the required configuration under this namespace:

```
<Config name="com.cronos.onlinereview.ajax.factory">
      <!-- defining the "SetScorecardStatus" handler concrete
      implementation -->
      <Property name="SetScorecardStatus">
            <Property name="type">
                  <Value>com.cronos.onlinereview.ajax.handlers.SetScorec
                  ardStatusHandler</Value>
            </Property>
      </Property>

      <!-- defining the "LoadTimelineTemplate" handler concrete
      implementation -->
      <Property name="LoadTimelineTemplate">
            <Property name="type">
                  <Value>com.cronos.onlinereview.ajax.handlers.LoadTimel
                  ineTemplateHandler</Value>
            </Property>
      </Property>

      <!-- defining the "SetTimelineNotification" handler concrete
      implementation -->
      <Property name="SetTimelineNotification">
            <Property name="type">
                  <Value>com.cronos.onlinereview.ajax.handlers.SetTimeli
                  neNotificationHandler</Value>
            </Property>
      </Property>

      <!-- defining the "PlaceAppeal" handler concrete implementation-->
      <Property name="PlaceAppeal">
            <Property name="type">
                  <Value>com.cronos.onlinereview.ajax.handlers.PlaceAppe
                  alHandler</Value>
            </Property>
      </Property>

      <!--defining the "ResolveAppeal" handler concrete
      implementation -->
```

```
        <Property name="ResolveAppeal">
              <Property name="type">
                    <Value>com.cronos.onlinereview.ajax.handlers.ResolveAp
                    pealHandler</Value>
              </Property>
        </Property>

        <!-- defining the resource manager concrete implementation -->
        <Property name="com/topcoder/management/resource/ResourceManager">
              <Property name="type">
                    <Value>com.topcoder.management.resource.DefaultResourc
                    eManager</Value>
              </Property>
        </Property>

        <!-- the rest of the manager classes must be defined here -->
              .....manager classes concrete implementation definition.....
</Config>
```

3.2.3 Ajax operation parameters

3.2.3.1 Set scorecard status

The request type is: *"SetScorecardStatus"*.

The parameters are:

Parameter	**Description**	**Values**
ScorecardId	The ID of the scorecard to update its status.	A long number.
Status	The status value.	"Active" / "Inactive"

The return value is: N/A.

3.2.3.2 Load timeline template

The request type is: *"LoadTimelineTemplate"*.

The parameters are:

Parameter	**Description**	**Values**
TemplateName	The name of the template used	A not null and not empty

	to generate the timeline data.	string
StartDate	An optional start date, from where to calculate the project phases dates. [Optional]	A date formatted as described in the algorithm section.

The return value is: XML representation of the calculated timeline data.

3.2.3.3 Set Timeline notification

The request type is: *"SetTimelineNotification"*.

The parameters are:

Parameter	**Description**	**Values**
ProjectId	The ID of the project which the notifications concerns.	A long number.
Status	The status value.	"On" / "Off"

The return value is: N/A.

3.2.3.4 Place appeal

The request type is: *"PlaceAppeal"*.

The parameters are:

Parameter	**Description**	**Values**
ReviewId	The ID of the review concerned by the appeal.	A long number.
ItemId	The ID of the review item concerned by the appeal.	A long number.
Text	The appeal text.	Any string

The return value is: N/A.

3.2.3.5 Resolve appeal

The request type is: *"ResolveAppeal"*.

The parameters are:

Parameter	**Description**	**Values**
ReviewId	The ID of the review concerned by the appeal	A long number.

	response.	
ItemId	The ID of the review item concerned by the appeal response.	A long number.
Status	The success of failure statuses.	“Succeeded” / “Failed”
Answer	The answer of the appeal response.	Any string
Text	The text of the appeal response.	Any string

The return value is: The new calculated score of the review.

3.2.4 Ajax response statuses

This table below lists all the Ajax response statuses.

Status	**Description**
Success	Returned when an operation was successful.
Login error	Returned when the user is not logged in.
Role error	Returned when the user has not the right to perform the operation.
Phase error	Returned when the phase constraint is not fulfilled in order to perform the operation.
Business error	Returned when the business tier failed to execute an operation.
Invalid parameter error	When a request parameter is missing or has an invalid value.
Invalid scorecard error	When the scorecard was not found in the business tier.
Invalid review error	When the review was not found in the business tier.
Invalid item error	When the review item was not found in the business tier.
Invalid template name error	When the template was not found in the business tier.

3.3 Dependencies Configuration

Configure all the components defined in section 2.2 *“TopCoder Software Components”*.

Configure the *AjaxSupportServlet* in the web application.

4. Usage Notes

4.1 Required steps to test the component

- Extract the component distribution.
- Make sure you tomcat home and cactus tmp directory path doesn't contain space. The jaxp parser will fail in this situation when it reads the schema file. (if it has, you can just use "set tmp={some dir not has space in path}" in the console to avoid this.).
- I also provide a war that contains all the demos. You can simply put it into {TOMCAt_HOME}/webapps/ and then restart tomcat. The demo url is not very easy so I give a shortcut here.

  http://localhost:8080/online_review_ajax_support-test/demoX.html (where X is from 1 to 6)
- Follow Dependencies Configuration.
- Execute 'ant test' within the directory that the distribution was extracted to.

4.2 Required steps to use the component

Follow dependencies configuration, and then deploy the web application.

4.3 Demo

This demo will demonstrate the usage of this component in order to execute the *"Load Timeline Template"* operation.

This is a snippet of the client side script:

```
// creating an XMLHttpRequest
var request;

// creating the request object in other browsers
if (window.XMLHttpRequest){
      request=new XMLHttpRequest();
}
// creating the request object in IE
else if (window.ActiveXObject){
      request=new ActiveXObject("Microsoft.XMLHTTP");
}
```

Configure the *XMLHttpRequest* object with the *onReadyStateChange* message handler function.

```
request.onreadystatechange= onReadyStateChange;
```

Create the XML document string which represents an Ajax request with type *"LoadTimelineTemplate"*.

The timeline we want to load is for a design project, so we use the *"Design"* template name, we use the default date.

Create the Ajax request XML document like this:

```
var requestString = '<?xml version="1.0" ?> '+
```

```
      '<request type="LoadTimelineTemplate">'+
            '<parameters>'+
                  '<parameter name="TemplateName">Design</parameter>'+
            '</parameters>'+
      '</request>';
```

Initialize the XMLHttpRequest object with the *"POST"* method and the Servlet URL

```
request.open("POST",servletUrl,true);
```

Set the content type to *"text/xml"*, since the request is sending an XML document.

```
request.setRequestHeader('Content-Type', 'text/xml');
```

Now, send the Ajax request.

```
request.send(requestString);
```

The *LoadTimelineTemplate* Ajax handler will process the request and return this XML document.

```
<?xml version="1.0" ?>
  <response type="LoadTimelineTemplate">
    <result status="Success">
      <timeline>
            <phases>
                  <phase type="Registration">
                        <start-date>03.30.2006 9:00 AM</start-date>
                        <end-date>04.02.2006 9:00 AM</end-date>
                        <!—- this phase length represents 72hrs in
                        milliseconds -->
                        <length>259200000</length>
                  </phase>

                  <phase type="Submission">
                        <start-date>04.02.2006 9:00 AM</start-date>
                        <end-date>04.07.2006 9:00 AM</end-date>
                        <!—- this phase length represents 121hrs in
                        milliseconds -->
                        <length>435600000</length>
                  </phase>

                  <!—- other phases were omitted -->
                  ............

            </phases>
      </timeline>
   </result>
</response>
```

The request object will receive the Ajax response in the *onReadyStateChange* function

```
function onReadyStateChange(){
      if( this.readyState == 4 && this.status == 200){
```

```
            // the Ajax response must be obtained as a DOM document
            var responseDom= this.responseXML.documentElement;

            // the Ajax request content could be accessd now
            // using the responseDom object
            ............
      }
}
```

4.3.1 *Sample XML for the "Set Scorecard Status" operation*

A global manager wants to activate a scorecard; the Ajax request XML would be like this:

```
<?xml version="1.0" ?>
<request type="SetScorecardStatus">
      <parameters>
            <!-- the scorecard ID parameter -->
            <parameter name="ScorecardId">45</parameter>

            <!-- the scorecard status parameter put to "Active" -->
            <parameter name="Status">Active</parameter>
      </parameters>
</request>
```

The component will process the request; the Ajax response XML would be like this:

```
<?xml version="1.0" ?>
<response type="SetScorecardStatus">
      <!-- the status is "Success", and there is no result data-->
      <result status="Success" />
</response>
```

4.3.2 *Sample XML for the "Set Timeline Notification" operation*

A user wants to enable receiving timeline notification on a project; the Ajax request XML would be like this:

```
<?xml version="1.0" ?>
<request type="SetTimelineNotification">
      <parameters>
            <!-- the project ID parameter -->
            <parameter name="ProjectId">51</parameter>

            <!-- the notification status parameter put to "On" -->
            <parameter name="Status">On</parameter>
      </parameters>
</request>
```

The component will process the request; the Ajax response XML would be like this:

```
<?xml version="1.0" ?>
<response type="SetTimelineNotification">
      <!-- the status is "Success", and there is no result data-->
```

```
      <result status="Success" />
</response>
```

4.3.3 *Sample XML for the "Place Appeal" operation*

A submitter decided to appeal an item in the review scorecard, Ajax request XML would be like this:

```
<?xml version="1.0" ?>
<request type="PlaceAppeal">
      <parameters>
            <!-- the review ID parameter -->
            <parameter name="ReviewId">496</parameter>

            <!-- the item ID parameter -->
            <parameter name="ItemId">496</parameter>

            <!-- the appeal text parameter -->
            <parameter name="Text">I have described this issue in the CS
            please refer to it</parameter>
      </parameters>
</request>
```

The component will process the request; the Ajax response XML would be like this:

```
<?xml version="1.0" ?>
<response type="PlaceAppeal">
      <!-- the status is "Success", and there is no result data-->
      <result status="Success" />
</response>
```

4.3.4 *Sample XML for the "Resolve Appeal" operation*

A reviewer decided to increase the score of an item, the Ajax request XML would be like this:

```
<?xml version="1.0" ?>
<request type="ResolveAppeal">
      <parameters>
            <!-- the review ID parameter -->
            <parameter name="ReviewId">496</parameter>

            <!-- the item ID parameter -->
            <parameter name="ItemId">496</parameter>

            <!-- the notification status parameter put to "Succeeded"-->
            <parameter name="Status">Succeeded</parameter>

            <!-- the answer parameter here put to 4 -->
            <parameter name="Answer">4</parameter>

            <!-- the appeal response text parameter -->
```

```
            <parameter name="Text">Yes, I will raise the
            score</parameter>
      </parameters>
</request>
```

The component will process the request and recalculate the score and return it; the Ajax response XML would be like this:

```
<?xml version="1.0" ?>
<response type="ResolveAppeal">
      <!-- the status is "Success" and the data is the review score -->
      <result status="Success">0.982144</result>
</response>
```

4.3.5 Sample XML for the unsuccessful operations

When the component fails to process an operation due to some reason the Ajax response XML will have an error status with optionally a descriptive message about the error.

For instance, a user wants to set the scorecard status to “Active” and he doesn’t have the global manager role, so the component will detect this error and return the following Ajax response XML:

```
<?xml version="1.0" ?>
<response type="SetScorecardStatus">
      <!-- the status is "Role error" and the data is the error message
      -->
      <result status="Role error">The user must have the global manager
      role to be able to perform this operation</result>
</response>
```

5. Future Enhancements

This component is extensible, new Ajax request handlers could be written in order to support new operations.