Billing Cost Struts Actions 1.0 Component Specification

1. Design

The main goal of this project is to deliver an efficient application for automatically importing hundreds of payment/fees records from PACTS to QuickBooks application. Parts of this goal will be filtering data records (like by date/time range, by customer, by projects, etc.), viewing history of imports and audit data.

This component is responsible for providing billing cost struts actions.

1.1 Design Patterns

1.1.1 Strategy

BaseAction uses BillingCostAuditService implementations which are pluggable.
BaseBillingCostReportAction uses BillingCostDataService implementations which are pluggable.

In addition, the actions are also used strategically by the Struts framework.

1.1.2 DAO

BillingCostAuditService and BillingCostDataService can be viewed as DAO used by this component.

1.1.3 DTO

The AccountingAuditRecord, BillingCostExportDetail, BillingCostExport, BillingCostReportEntry, PaymentIdentifier, and all the criteria classes are DTO used by this component.

1.1.4 MVC

Actions can be treated like the Controller part of the MVC pattern.

1.1.5 IoC

The configuration is done through Spring injection. Therefore the Inversion of Control (IoC) pattern is used.

1.2 Industry Standards

XML
JSP
HTML
HTTP
OGNL

1.3 Required Algorithms

Please refer to TCUML method doc for details not listed below.

1.3.1 Logging

The component will log activity and exceptions using the Logging Wrapper in struts actions.

It will log errors at Error level, and method entry/exit information at DEBUG level.

Specifically, logging will be performed as follows, if logging is turned on.

- Method entrance and exit will be logged with DEBUG level.
  - Entrance format: [Entering method {*className.methodName}*]
  - Exit format: [Exiting method {*className.methodName}*]. Only do this if there are no exceptions.
- Method request and response parameters will be logged with DEBUG level
  - Format for request parameters: [Input parameters[{request_parameter_name_1}:{ request_parameter_value_1}, {request_parameter_name_2}:{ request_parameter_value_2}, etc.)]]

    - Format for the response: [Output parameter {response_value}] . Only do this if there are no exceptions and the return value is not void.
    - If a request or response parameter is complex, use its toString() method. If that is not implemented, then print its value using the same kind of name:value notation as above. If a child parameter is also complex, repeat this process recursively.
- All exceptions will be logged at ERROR level, and automatically log inner exceptions as well.
    - Format: Simply log the text of exception: [Error in method {*className.methodName}*: Details {*error details*}]

In general, the order of the logging in a method should be as follows:

1. Method entry
2. Log method entry
3. Log method input parameters
4. If error occurs, log it and skip to step7
5. Log method exit
6. If not void, log method output value
7. Method exit

The toJSONString() method of the entities should used for the logging.

1.3.2 Error Handling

The actions catch any error, wrap as Exception, and throw it. This exception will be caught and the user will be redirected to some error page. This is out of scope because it's handled by the struts configuration of the overall web application. This component only needs to throw exception without worrying about redirecting the user to the error page.

1.3.3 JSP

When JSPs are developed later, their interactions with the actions can be as follows (the JSP file names are just example):
billingCostReport.jsp – this will display the billing cost report using BillingCostReportAction, and export the report using BillingCostReportQuickBooksExportAction.
billingCostExportHistory.jsp – this will display the billing cost export history using BillingCostExportHistoryAction.
billingCostExportDetails.jsp – this will display the details of a billing cost export using BillingCostExportDetailsAction.
auditHistory.jsp – this will display the accounting audit records using AuditHistoryAction.

For the input and output of each action, please see TCUML.

1.3.4 Validation

Some complex validation is performed programmatically in BillingCostReportQuickBooksExportAction. Other than that, the rest of the validation is done completely in XML.

The developers should write the Struts XML validation files according to the validation rules given below. See here for instructions on writing the XML files and here for the validators to use. The error message can be hard-coded text, rather than using resource files.

1.3.4.1 pageNumber and pageSize

All actions having these two parameters should have non-negative values for them. This can be done using the int validator. They are also mandatory parameters, so the required validator should be used.

```
<field name="pageNumber">
        <field-validator type="int">
                <param name="min">0</param>
                <message><![CDATA[ page number must be non-negative]]></message>
        </field-validator>
</field>
<field name="pageSize">
```

```
        <field-validator type="int">
                <param name="min">0</param>
                <message><![CDATA[ page size must be non-negative]]></message>
        </field-validator>
</field>
```

1.3.4.2 startDate and endDate

The startDate must not be larger than the endDate for all criteria classes. This can be achieved using fieldexpression validator and OGNL expression like this:

```
<field name="criteria.startDate">
        <field-validator type="date">
                <message>Start?Date?is required.</message>
        </field-validator>
</field>
<field name="criteria.endDate">
        <field-validator type="date">
                <message>End Date is required.</message>
        </field-validator>
</field>
<validator type="fieldexpression">
       <param name="expression"><![CDATA[ criteria.startDate <= criteria.endDate]]></param>
       <message>Start?Date?must?not?be?larger?than?End?Date.</message>
 </validator>
```

1.3.4.3 BillingCostExportHistoryAction's BillingCostExportHistoryCriteria

The BillingCostExportHistoryCriteria.accountantName must not be longer than 50 chars. This can be done using the stringlength validator.

```
<field name="criteria.accountantName ">
        <field-validator type="stringlength">
                <param name="maxLength">50</param>
                <message><![CDATA[accountant name must not be longer than 50
chars]]></message>
        </field-validator>
</field>
```

1.3.4.4 AuditHistoryAction's AccountingAuditRecordCriteria

The AccountingAuditRecordCriteria.userName must not be longer than 50 chars. This can be done using the stringlength validator which is similar as above.

1.3.5 @PostConstruct

The method that is marked with <<@PostConstruct>> in TCUML should be set as the value of the "init-method" attribute in the bean declaration of the Spring application context file. These methods don't need to be added with the javax.annotation.PostConstruct.

1.4 Component Class Overview

com.topcoder.accounting.action:

BillingCostExportDetailsAction:

This action is responsible for getting the billing cost export details of a particular billing cost export. It supports pagination.

Thread Safety:
This class is not thread-safe because it's mutable. However, dedicated instance of struts action will be created by the struts framework to process the user request, so the struts actions don't need to be thread-safe.

BillingCostExportHistoryAction:

This action is responsible for getting a list of billing cost export records by searching criteria given by BillingCostExportHistoryCriteria. It supports pagination.

Thread Safety:

This class is not thread-safe because it's mutable. However, dedicated instance of struts action will be created by the struts framework to process the user request, so the struts actions don’t need to be thread-safe.

BaseAction:

This is the base class for all actions of this component. It extends ActionSupport, and implement ServletRequestAware to get access to the http request object. It defines a method for auditing that all subclasses can use. Its billingCostAuditService and servletRequest are also available for the subclasses.

Thread Safety:
This class is not thread-safe because it's mutable. However, dedicated instance of struts action will be created by the struts framework to process the user request, so the struts actions don’t need to be thread-safe.

BaseBillingCostAuditAction:

This is the base class for actions that are used to show auditing information, including billing cost export history/details, and auditing records. It simply has page number and page size properties that subclasses can use to support pagination.

Thread Safety:
This class is not thread-safe because it's mutable. However, dedicated instance of struts action will be created by the struts framework to process the user request, so the struts actions don’t need to be thread-safe.

AuditHistoryAction:

This action is responsible for getting a list of accounting audit records by searching criteria given by AccountingAuditRecordCriteria. It supports pagination.

Thread Safety:
This class is not thread-safe because it's mutable. However, dedicated instance of struts action will be created by the struts framework to process the user request, so the struts actions don’t need to be thread-safe.

BillingCostReportAction:

This action is responsible for getting a list of billing cost report entries by searching criteria given by BillingCostReportCriteria. It supports pagination.

Thread Safety:
This class is not thread-safe because it's mutable. However, dedicated instance of struts action will be created by the struts framework to process the user request, so the struts actions don’t need to be thread-safe.

BillingCostReportQuickBooksExportAction:

This action is responsible for exporting a list of billing cost report entries to Quick Books. The list of billing cost report entries to export are identified by a list of PaymentIdentifier object.

Thread Safety:
This class is not thread-safe because it's mutable. However, dedicated instance of struts action will be created by the struts framework to process the user request, so the struts actions don’t need to be thread-safe.

BaseBillingCostReportAction:

This is the base class for actions about billing cost report, including getting the report and exporting the report. It simply has a BillingCostDataService that subclasses can use.

Thread Safety:
This class is not thread-safe because it's mutable. However, dedicated instance of struts action will be created by the struts framework to process the user request, so the struts actions don't need to be thread-safe.

com.topcoder.accounting.action.converter:

DateConverter:

This is a custom DefaultTypeConverter subclass that is responsible for conversion between String and Date, according to the date format string configured as servlet parameter. It is used by Struts2 to properly convert the date string into Date object.

Thread Safety:
This class is thread-safe because it's immutable assuming input parameters are used safely by the caller, and because the super class is thread-safe

1.5 Component Exception Definitions

1.5.1 Custom exceptions

BillingCostActionConfigurationException:

This is thrown if any configuration error occurs for the actions of this component.

Thread Safety:
This class is not thread-safe because the base class is not thread-safe.

DateConversionException:

This is thrown if any error occurs during conversion between date and string. It's thrown by DateConverter.

Thread Safety:
This class is not thread-safe because the base class is not thread-safe.

1.5.2 System exceptions

IllegalArgumentException:

This will be used to signal that an input parameter to the component is illegal.

1.6 Thread Safety

Although individual classes in this component are not thread-safe as per 1.4, this component is effectively thread-safe to use under Struts framework because dedicated instances of struts actions will be created by the Struts framework to process each user request cause the actions should be configured with "scope="request" in Spring configuration file, and because the struts action instances will not be changed after initialization, and finally the BillingCostAuditService and BillingCostDataService used by the actions are all thread-safe to use.

2. Environment Requirements

2.1 Environment

Java 1.5

Informix 11.5

JBoss 4.0.2

2.2 TopCoder Software Components

Logging Wrapper 2.0 – provides logging service

Custom Component:

Billing Cost Services 1.0 – provides BillingCostAuditService and BillingCostDataService.

Cockpit TC Application – this component will be eventually integrated into the web application.

2.3 Third Party Components

Spring 2.5.6 (http://www.springsource.org/download)

Struts 2.2.3 (http://struts.apache.org/download.cgi)

3. Installation and Configuration

3.1 Configuration Parameters

Configuration for BaseAction:

Name	Description	Value
billingCostAuditService	The BillingCostAuditService used for auditing purpose and for the subclasses to use.	com.topcoder.accounting.service.BillingCostAuditService instance. It cannot be null. Required.
action	The action type of this action. This is configured for each subclass for auditing purpose.	java.lang.String instance. It cannot be null or empty. Required.

Configuration for BaseBillingCostReportAction:

Name	Description	Value
billingCostDataService	The BillingCostDataService used by the subclasses to get and export billing cost report entries.	com.topcoder.accounting.service.BillingCostDataService instance. It cannot be null. Required.

Please see the configuration of its base class BaseAction for more configuration parameters.

Configuration for BillingCostReportAction:

Name	Description	Value
defaultDuration	The default duration in milliseconds between start date and end date of the search criteria when start date is not given.	long. It must be non-negative. Required.

Please see the configuration of its base class BaseBillingCostReportAction for more configuration parameters.

Configuration for AuditHistoryAction:
Please see the configuration of its base class BaseBillingCostAuditAction for its configuration parameters.

Configuration for BillingCostExportDetailsAction:
Please see the configuration of its base class BaseBillingCostAuditAction for its configuration parameters.

Configuration for BillingCostExportHistoryAction:
Please see the configuration of its base class BaseBillingCostAuditAction for its configuration parameters.

Configuration for BillingCostReportQuickBooksExportAction:
Please see the configuration of its base class BaseBillingCostReportAction for its configuration parameters.

Configuration for DateConverter:

Name	Description	Value

datePattern	The date format string used to parse string into date.	String. Non-empty. It's configured in web.xml in <context-param> as the serlvet parameter of the Struts servlet.

In addition, DateConverter must be plugged into the Struts framework by setting java.util.Date= com.topcoder.accounting.action.converter.DateConverter in the Struts configuration file 'xwork-conversion.properties'.

The above tables' configuration should be done in Spring XML except datePattern of DateConverter.

3.2 Dependencies Configuration

Logging Wrapper should be configured properly. The depended back-end services should be configured properly.

3.3 Package Structure

com.topcoder.accounting.action
com.topcoder.accounting.action.converter

4. Usage Notes

4.1 Required steps to test the component

- Extract the component distribution.
- Update the build-dependencies.xml to add struts and spring dependencies.
- Execute 'ant test' within the directory that the distribution was extracted to.

4.2 Required steps to use the component

Please see the demo.

4.3 Demo

A sample Spring configuration file "applicationContext.xml" is attached for reference only. Since Struts2 does all the work there is nothing to show in demo in fact. Here a sample JSP for getting audit history and its interaction with AuditHistoryAction is shown:

```
<%...@ page language="java" contentType="text/html>
<%...@ taglib prefix="s" uri="/struts-tags" %>
<html>
  <head>
    <title></title>
  </head>
  <body>
     <s:form action="getAuditHistory">
       <s:textfield name="criteria.startDate" key="startDate"></s:textfield>
       <s:textfield name="criteria.endDate" key="endDate"></s:textfield>
       <s:textfield name="criteria.action" key="action"></s:textfield>
       <s:textfield name="criteria.userName" key="userName"></s:textfield>
        <s:textfield name="pageNumber" key="pageNumber"></s:textfield>
        <s:textfield name="pageSize" key="pageSize"></s:textfield>
       <s:submit value="submit"></s:submit>
     </s:form>
  </body>
</html>
```

If pass the validation then success page will show the record.

```
<%@page language="java" contentType="text/html" %>
<%@taglib prefix="s" uri="/struts-tags" %>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title></title>
</head>
<body>
   <!-- here a "for-loop" can be used to display the search result after the user clicks
   'submit', which can be accessed as 'accountingAuditRecords' property.-->
```

```
  <s:iterator value="accountingAuditRecords.records" id="record">
       Action : <s:property value="#record.action"/> <br/>
       Username : <s:property value="#record.userName"/> <br/>
       Timestamp : <s:property value="#record.timestamp"/> <br/>
  </s:iterator>
</body>
</html>
```

Suppose that the user inputs “getBillingCostExportHistory” for ‘action’ field, ‘2011/08/30’ for ‘startDate’, and ‘2011/08/31’ for ‘endDate’, ‘pageNumber=2’, ‘pageSize=10’, and assume that the ‘datePattern’ is configured as ‘yyyy/MM/dd’ in web.xml, and assume that when the user clicks ‘submit’, AuditHistoryAction handles the request. AuditHistoryAction.execute() will retrieve the 2nd page (each page showing 10 records) of all audit records for access billing cost export history between 2011/08/30 and 2011/08/31 by all users. This information will be put into ‘accountingAuditRecords’ property, and the JSP can display it easily with Struts tags. This access is also audited as action ‘getAuditHistory’, according to the sample Spring configuration file.

Other actions are used in a similar manner.

5. Future Enhancements

None