Billing Cost QuickBooks Service 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 the QuickBooks Web Connector web service to import invoices into QuickBooks.

1.1 Design Patterns

Strategy pattern – BillingCostAuditService and Log are used by QBWCImportInvoicesServiceSkeleton as a strategy. QBWCImportInvoicesServiceSkeleton is also a strategy for QBWebConnectorSvcSoap.

Proxy pattern – This component will be deployed as web service, hence the clients will use the proxy to access the web service.

DTO pattern – The method arguments of QBWCImportInvoicesServiceSkeleton are the DTO objects.

1.2 Industry Standards

JavaBeans, IoC, Web Service

1.3 Required Algorithms

1.3.1 Logging

Logging will be done using Logging Wrapper. Each exception will be logged at ERROR level. There will be logging of method entry/exit and parameter input/output at the DEBUG level.

The logging of input/output information requires detailed printing of entity data. In the past, this was done by ad-hoc development decisions about where to put that functionality. Some put it in helper classes, some put it in the toString method. In cases where the logging format was not specified, different components relied on their own printing standards.

To this end, this project will enforce the following standard: All entities will provide a JSON version of their data, to be supplied in a toJSONString():String method. This will standardize the logging of input and out information of entities.

1.3.2 Auditting

In all the methods of the QBWCImportInvoicesServiceSkeleton except the set/get injected methods and the default constructor, the auditing should be performed. BillingCostAuditService. +auditAccountingAction(record:AccountingAuditRecord):void should be called. AccountingAuditRecord.actionName should be the method name called, and AccountingAuditRecord.userName should be “QuickBooks Web Service”, and the timestamp should be the current time.

1.4 Component Class Overview

QBWCImportInvoicesServiceSkeleton

QBWCImportInvoicesServiceSkeleton represents a web service used to authenicate user, send/recived the xml request/response.

The other methods are left the same as the proof-assembly.

It will also log the errors and events(see the CS for detail).

It's mutable and not thread safe, but when it is used as a singleton instance as the webserice, it's thread safe.

1.5 Component Exception Definitions

None.

1.6 Thread Safety

This component is thread safe when used correctly.

QBWCImportInvoicesServiceSkeleton is mutable and not thread safe. In the apache AXIS 2 framework, it acts a state-full webservice, and the framework will guarantee that it's used in the thread safe model.

2. Environment Requirements

2.1 Environment

Development language: Java 1.5
Compile Target: Java 1.5, J2EE 1.5
IoC
JSON
Spring
XML

2.2 TopCoder Software Components

- **Logging Wrapper 2.0** – provides the logging utility.
- **Hash Utility 1.0** – provides the hash utility.
- **Billing Cost Services 1.0** – provides the services to manage the billing cost data.

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

- Spring 2.5.6: http://www.springsource.org/
- Apache Axis2 1.5.5 : http://axis.apache.org/axis2/java/core/

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.topcoder.accounting

3.2 Configuration Parameters

All the fields are injected, please see the class diagram for detail.

3.3 Dependencies Configuration

Please see docs of dependency components to configure them properly.

4. Usage Notes

4.1 Required steps to test the component

- Extract the component distribution.
- Download Axis2 1.5.5 WAR Distribution from http://archive.apache.org/dist/axis/axis2/java/core/1.5.5/axis2-1.5.5-war.zip and install it into your J2EE server.
- Update build-dependencies.xml, change the value of property “deploy_target” to the installation of your Axis2 1.5.5.
- Execute ‘ant deploy_demo’ to deploy the web service.
- Open *your_installation_of_axis2*/WEB-INF/web.xml, and add the following line:

```
<listener>
    <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>
</listener>
<context-param>
    <param-name>contextConfigLocation</param-name>
    <param-value>/WEB-INF/applicationContext.xml</param-value>
</context-param>
```

- Update test_files/demo.properties to setup the right URL of the deployed web service.
- Start your J2EE server.
- 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

4.3.1 Usage demo

.

Apache Axis includes the wsdl2java tool for creating a complete java source framework to implement a web service based on a WSDL. Again, by default WSDL2Java generates a Java Proxy class to communicate with a web service implementing the given WSDL, but the -server-side and -skeletonDeploy flags can be used to create the server-side framework, including an Axis Web Services Deployment Descriptor (wsdl) file:
java org.apache.axis.wsdl.WSDL2Java-server-side-skeletonDeploy http://developer.intuit.com/uploadedFiles/Support/QBWebConnectorSvc.wsdl

The way to access the web service proxy look like this:

```
// create web service stub
QBWebConnectorSvcStub stub = new QBWebConnectorSvcStub(LoadEndPointUrl());

// authenticate
Authenticate authenticate = new Authenticate();
AuthenticateResponse authenticateResponse = stub.authenticate(authenticate);

// send request xml
SendRequestXMLResponse sendRequestXMLResponse = stub.sendRequestXML(new SendRequestXML());
String response = sendRequestXMLResponse.getSendRequestXMLResult();

// receive response xml
ReceiveResponseXMLResponse receiveResponseXMLResponse = stub.receiveResponseXML(new
ReceiveResponseXML());
```

5. Future Enhancements

None