**FIRST CONNECT - BUSINESS LAYER**

**DEVELOPER GUIDE**

**Revision History**

|  |  |  |  |
| --- | --- | --- | --- |
| Version No | Date | Prepared by / Modified by | Significant Changes |
| 1.0 | 29-Apr-14 | Anitha Ramaiah | Initial Version |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

**Table of Contents**

Contents

[1. Getting Access 6](#_Toc389140537)

[1.1 Share Folder 6](#_Toc389140538)

[1.2 SVN 6](#_Toc389140539)

[2. Environment Setup 6](#_Toc389140540)

[2.1 JDK 6](#_Toc389140542)

[2.1.1 Prerequisites 6](#_Toc389140543)

[2.1.2 Configuration 6](#_Toc389140544)

[2.2 IDE 6](#_Toc389140545)

[2.2.1 Prerequisites 6](#_Toc389140546)

[2.2.2 Installation 7](#_Toc389140547)

[2.2.3 Configuration 7](#_Toc389140548)

[2.2.4 Verification 10](#_Toc389140549)

[2.3 Application Server 11](#_Toc389140550)

[2.3.1 Prerequisites 11](#_Toc389140551)

[2.3.2 Installation 11](#_Toc389140552)

[2.3.3 SSL Key Store Generation 11](#_Toc389140553)

[2.3.4 Configuration 12](#_Toc389140554)

[2.3.5 Verification 12](#_Toc389140555)

[2.4 Build Management 12](#_Toc389140556)

[2.4.1 Prerequisites 12](#_Toc389140557)

[2.4.2 Installation 12](#_Toc389140558)

[2.4.3 Configuration 14](#_Toc389140559)

[2.5 Version Control System 15](#_Toc389140560)

[2.5.1 Prerequisites 15](#_Toc389140561)

[2.5.2 Installation 15](#_Toc389140562)

[2.5.3 Configuration 15](#_Toc389140563)

[2.6 Web Service Framework 17](#_Toc389140564)

[2.6.1 Prerequisites 17](#_Toc389140565)

[2.6.2 Installation 17](#_Toc389140566)

[3. Business Layer Implementation 17](#_Toc389140567)

[3.1 Prerequisites 17](#_Toc389140570)

[3.1.1 Identify Services 17](#_Toc389140571)

[3.1.2 Identify API 17](#_Toc389140572)

[3.1.3 Identify Request-Response 18](#_Toc389140573)

[3.1.4 Identify Projects 18](#_Toc389140574)

[3.1.5 Web Project Creation 18](#_Toc389140575)

[3.1.6 Java Project Creation 20](#_Toc389140576)

[3.1.7 POM Creation 22](#_Toc389140577)

[3.1.8 Building Project 23](#_Toc389140578)

[3.2 Exposing Web Service 24](#_Toc389140579)

[3.2.1 Files Required 24](#_Toc389140580)

[3.2.2 Request-Response Creation 25](#_Toc389140581)

[3.2.3 End Point Creation 25](#_Toc389140582)

[3.2.4 End Point Implementation Creation 26](#_Toc389140583)

[3.2.5 Servlet Configurations 27](#_Toc389140584)

[3.2.6 Bean Configurations 28](#_Toc389140585)

[3.3 Business Logic Implementation 28](#_Toc389140586)

[3.3.1 Files Required 28](#_Toc389140587)

[3.3.2 Business Interface Creation 28](#_Toc389140588)

[3.3.3 Business Logic Implementation Creation 28](#_Toc389140589)

[3.3.4 Bean Configurations 29](#_Toc389140590)

[3.4 DAO Implementation 29](#_Toc389140591)

[3.4.1 Files Required 29](#_Toc389140592)

[3.4.2 DAO Interface Creation 29](#_Toc389140593)

[3.4.3 DAO Implementation Creation 30](#_Toc389140594)

[3.4.4 Transformers Creation 32](#_Toc389140595)

[3.4.5 Bean Creation 32](#_Toc389140596)

[3.5 ESB Integration 33](#_Toc389140597)

[3.5.1 Files Required 33](#_Toc389140598)

[3.5.2 Delegate Creation 33](#_Toc389140599)

[3.5.3 Configurations 34](#_Toc389140600)

[3.6 Framework Integration 34](#_Toc389140601)

[3.6.1 Logging 35](#_Toc389140602)

[3.6.2 Custom Logging 35](#_Toc389140603)

[3.6.3 Exception Handling 35](#_Toc389140604)

[3.6.4 WS Security 37](#_Toc389140605)

[3.6.5 Timeout 37](#_Toc389140606)

[3.7 Deployment 37](#_Toc389140607)

[3.8 Unit Testing 38](#_Toc389140608)

[3.9 Existing Project Structure 38](#_Toc389140609)

[4. Trouble Shooter 39](#_Toc389140610)

[4.1 Build Management 39](#_Toc389140611)

[4.2 Application Servers 39](#_Toc389140612)

[4.3 Deployment 40](#_Toc389140613)

# Getting Access

Access to the share folders and SVN repositories is required. Follow the instructions in the below sections to get the required access.

## Share Folder

Access to the following share folders is required. Get it from Archana ([Archana-r@hcl.com](mailto:Archana-r@hcl.com)).

1. 10.105.188.30
2. 10.105.130.10

## SVN

Access to the following repositories is required. Get it from Anitha Ramaiah ([Anitha.Ramaiah@hcl.com](mailto:Anitha.Ramaiah@hcl.com)).

1. http://10.105.188.28/fgbrepo
2. http://10.105.188.28/fgbdocs
3. <http://172.20.211.15/fbpsource> (or) http://10.105.67.138/fbpsource

# Environment Setup

Follow the instructions described in the below sections to install and configure the development Environment of First Connect. After executing each section return back to the appropriate section and continue the flow.



## JDK

### Prerequisites

1. Get the tarmac approval
2. Raise a remedy to install

### Configuration

Set java environment variables.

|  |
| --- |
| JAVA\_HOME=D:\FGBDEV\tools\ jdk1.6.0\_25  PATH=%PATH%;%JAVA\_HOME%\bin |

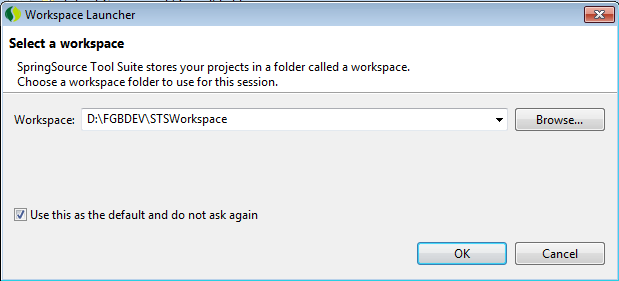
## IDE

### Prerequisites

1. Make sure you have access to server 10.105.188.30. If not, refer the section [Getting Access](#_Getting_Access_1).
2. Make sure Eclipse/STS/JBOSS developer studio exists in share folder 10.105.188.30/Softwares.
3. Get Tarmac approval for all the software to be used.
4. Ensure JDK is installed.

### Installation

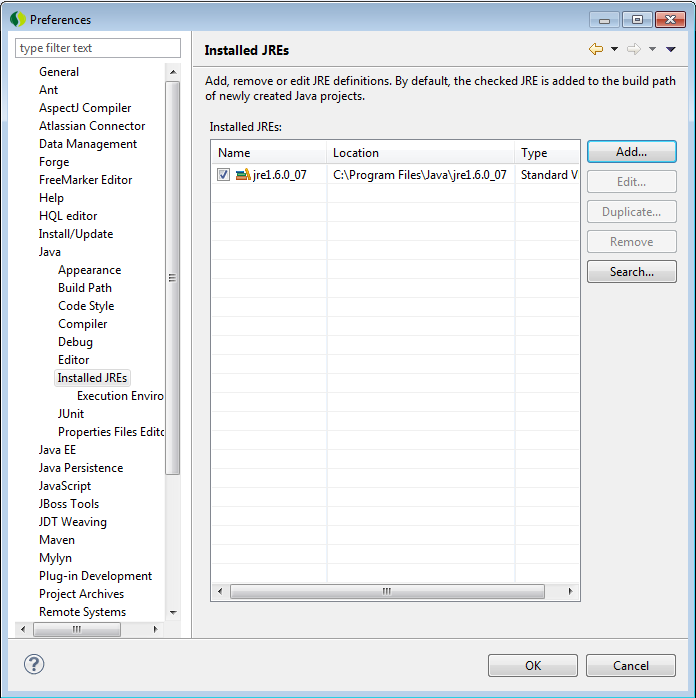
1. Copy the Eclipse/STS zip or JBOSS developer studio jar file from the share folder depending on your system configuration.
2. For Eclipse/STS, extract it to a folder in local drive (For JBOSS developer studio, install it by executing the installer).
3. Launch the application by opening executable file.
4. On prompt, specify the workspace.



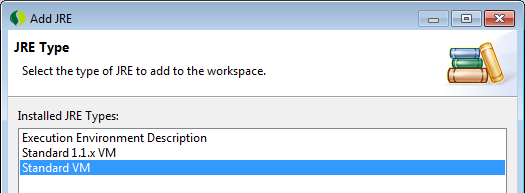
### Configuration

#### JRE Configuration

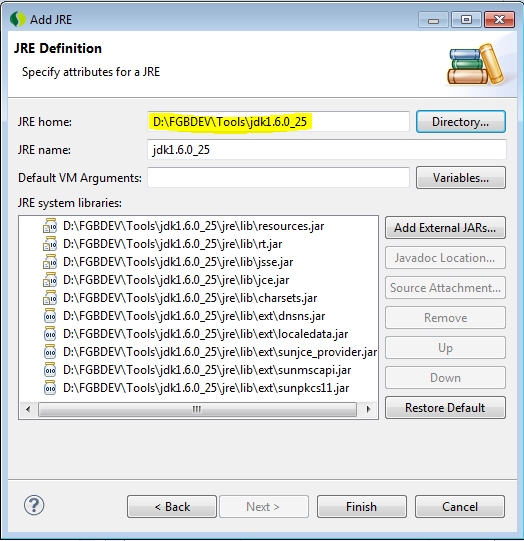
1. Open preferences window from menu windows-preferences.
2. Expand the option “java-installed JRE” from left navigation bar.



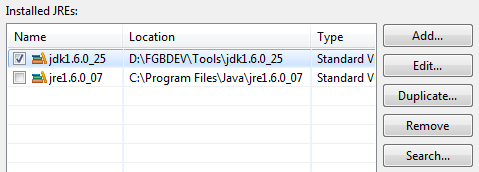
1. Click the add button to add the Standard VM.
2. Select the Standard VM option from the window as shown below.



1. Click next to proceed.
2. Specify the JRE Home as “D:\jdk1.6.0\_25”

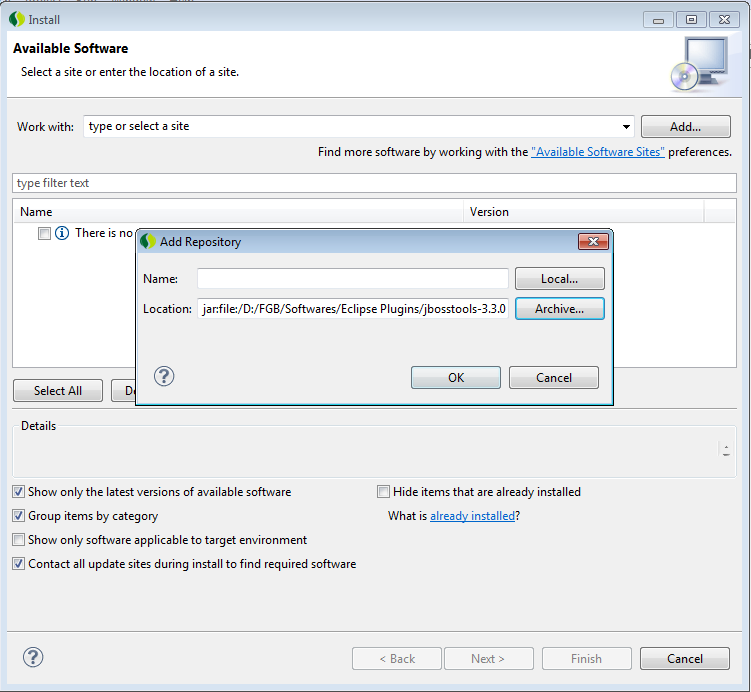


1. Click Finish.
2. Select the newly added JRE as shown below to make it the default JRE for the IDE.



#### JBoss Tools Plugins configuration

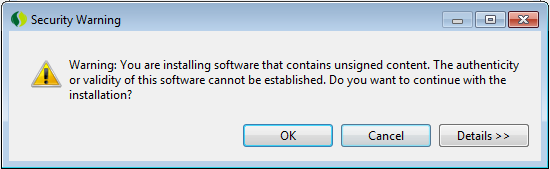
1. Select Help->Install New Software from menu option.
2. From the Install window, click the button Add.
3. From the Add Repository window click the button Archive and select the file jbosstools-3.3.0.M5.aggregate-Update-2011-12-14\_19-42-13-H61.zip from the share folder.



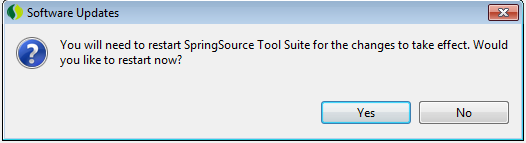
1. Click Ok to show the List of Available Softwares in a new Window.
2. Un Check the following Softwares from the list,

* JBoss GWT Integration
* JBoss BIRT Integration
* JBoss Cloud Development Tools

1. Accept the license terms and click ok to continue.
2. Click ok when the Security Warning pops up.

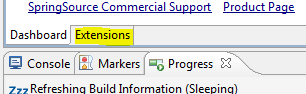


1. On Completion, it will prompt to reboot the workbench. Select the Yes button to restart STS.



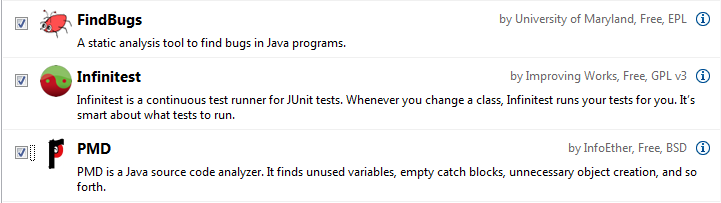
#### Additional Plugins Configuration

1. Select Help->Dashboard from the menu.
2. At the left bottom of the Dashboard window, select the Extensions tab as shown below,



1. Select the Following optional Plugins from the list and click Install button.

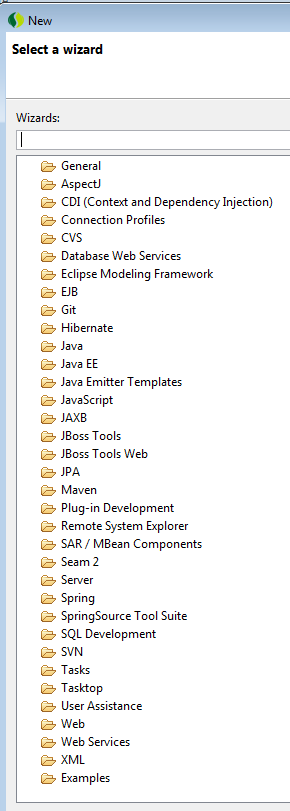
* FindBugs
* Infinitest
* PMD



1. Select all the softwares from the list that it shows and click next to proceed.
2. Accept the license and click next to proceed.
3. Click Ok button when the Security Warning window pops up and Click Yes when it prompts for reboot of STS.

### Verification

1. Start the IDE.
2. Ensure it doesn’t report any errors.
3. Verify whether the new Project window opens up a similar window with JBoss Tools and other plugin options as shown below (Rotate and see).



## Application Server

### Prerequisites

1. Make sure you have access to server 10.105.188.30. If not, refer the section [Getting Access](#_Getting_Access_1).
2. Make sure jboss-6.1.0.Final.zip exists in share folder 10.105.188.30/Softwares

### Installation

1. Copy the jboss-6.1.0.Final.zip file from the share folder.
2. Extract it to a folder in local drive.

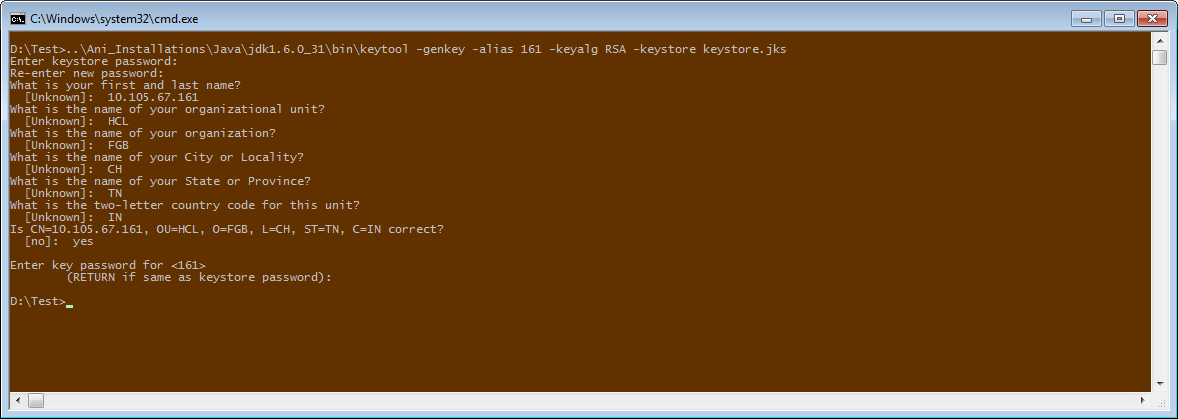
### SSL Key Store Generation

The below two sections can be omitted as of now. These are required when the services are exposed with SSL.

1. Create a new keystore using the Java supplied Keytool

**keytool -genkey -alias jboss -keyalg RSA -keystore keystore.jks**

1. Enter the following details CN, FN, City Code, Country code etc



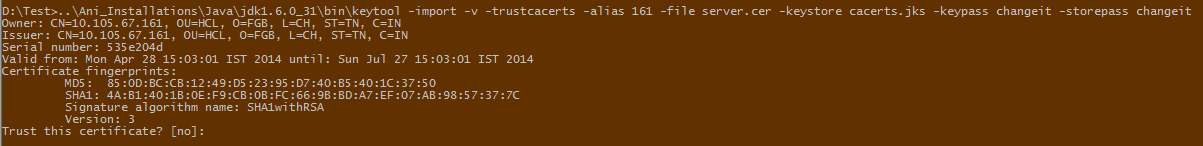
1. Export the generated server certificate from the keystore to server.cer.

**keytool –export –alias 161 -storepass changeit -file server.cer -keystore keystore.jks**



1. Create a trust-store file and add the server cert to it.

**keytool –import –v –trustcacerts -alias 161 -file server.cer -keystore cacerts.jks –keypass changeit -storepass changeit**



### Configuration

1. Edit the file server.xml as described below
2. Comment the following line.

<!--   
<Connector protocol="HTTP/1.1" SSLEnabled="true"   
port="${jboss.web.https.port}" address="${jboss.bind.address}"  
scheme="https" secure="true" clientAuth="false"  
keystoreFile="${jboss.server.home.dir}/conf/chap8.keystore"  
keystorePass="rmi+ssl" sslProtocol = "TLS" />  
-->

1. Add the following lines immediately after the line described above

<Connector protocol="HTTP/1.1" SSLEnabled="true"

port="${jboss.web.https.port}" address="${jboss.bind.address}"

scheme="https" secure="true" clientAuth="false"

keystoreFile="D:/CERTS/keystore.jks" keystorePass="changeit"

truststoreFile="D:/CERTS/cacerts.jks" truststorePass="changeit"

sslProtocol = "TLS" />

### Verification

1. Go to Command Prompt.
2. Start the server.
3. Hit the URL <https://localhost:8443> in browser. It should open the server page.

## Build Management

### Prerequisites

1. Make sure you have access to server 10.105.188.30. If not, refer the section [Getting Access](#_Getting_Access_1).
2. Make sure apache-maven-3.0.4.zip and repository.Final.zip exists in share folder 10.105.188.30/Softwares

### Installation

1. Copy the apache-maven-3.0.4.zip file from the share folder.
2. Extract it to a folder in local drive.
3. Copy the repository.Final.zip from the share folder.
4. Extract it to MAVEN\_HOME/respository folder.
5. Open Settings.xml from MAVEN\_HOME/conf.
6. Search for the word <localRepository>.
7. Comment it as below.

<!-- <localRepository>/path/to/local/repo</localRepository> -->

1. Add the below line to reflect your repository.

<localRepository>D:\Tools\apache-maven-3.0.4\repository </localRepository>

Before Change:

<localRepository>/path/to/local/repo</localRepository>

After Change:

<!-- <localRepository>/path/to/local/repo</localRepository> -->

<localRepository>D:\Tools\apache-maven-3.0.4\repository </localRepository>

1. Search for the word ‘proxies’.
2. Comment the section ‘proxy’ as below

<!--

<proxy>

<id>optional</id>

<active>true</active>

<protocol>http</protocol>

<username>proxyuser</username>

<password>proxypass</password>

<host>proxy.host.net</host>

<port>80</port>

<nonProxyHosts>local.net|some.host.com</nonProxyHosts>

</proxy>

-->

1. Add the below section inside the tag <proxies></proxies>.

<proxy>

<id>optional</id>

<active>true</active>

<protocol>http</protocol>

<username>anitha.ramaiah</username>

<password>MarPas@1</password>

<host>arihant-proxy</host>

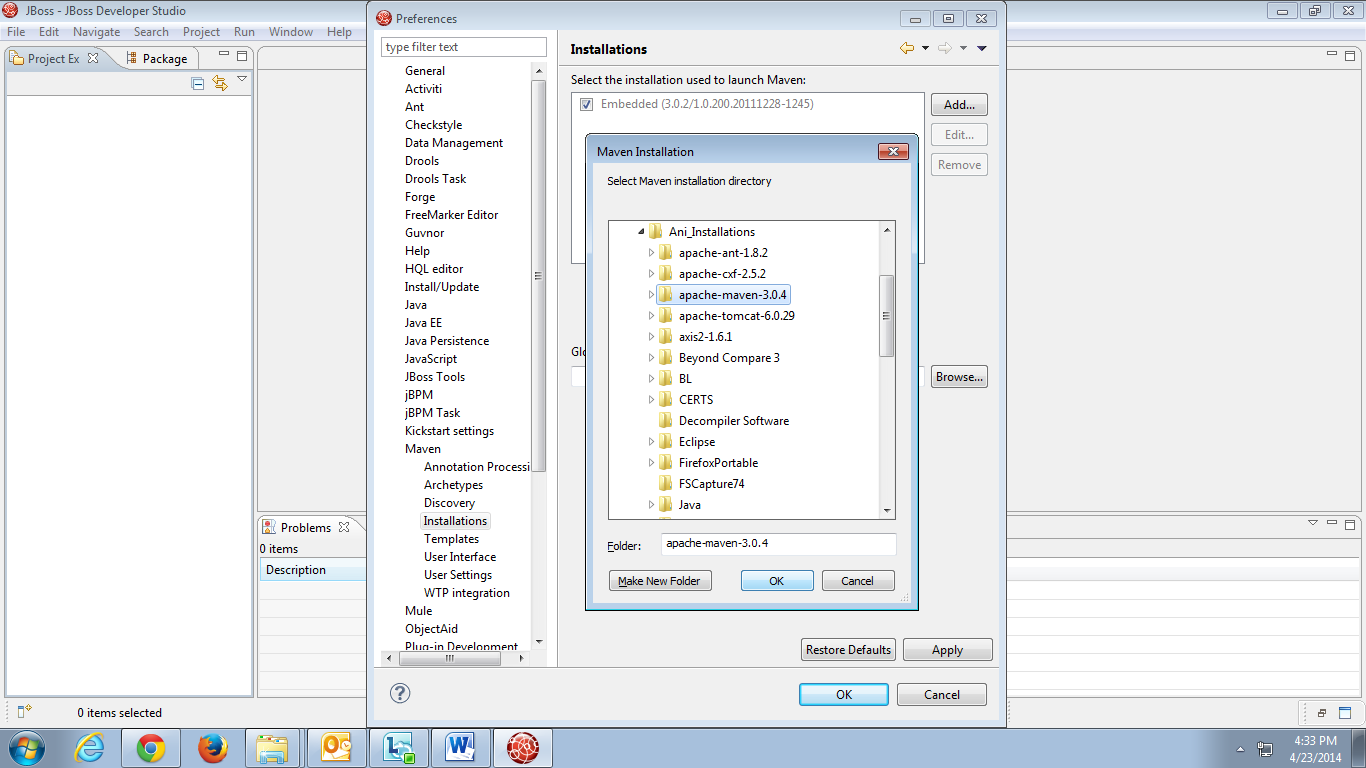
<port>80</port>

<nonProxyHosts>local.net|some.host.com</nonProxyHosts>

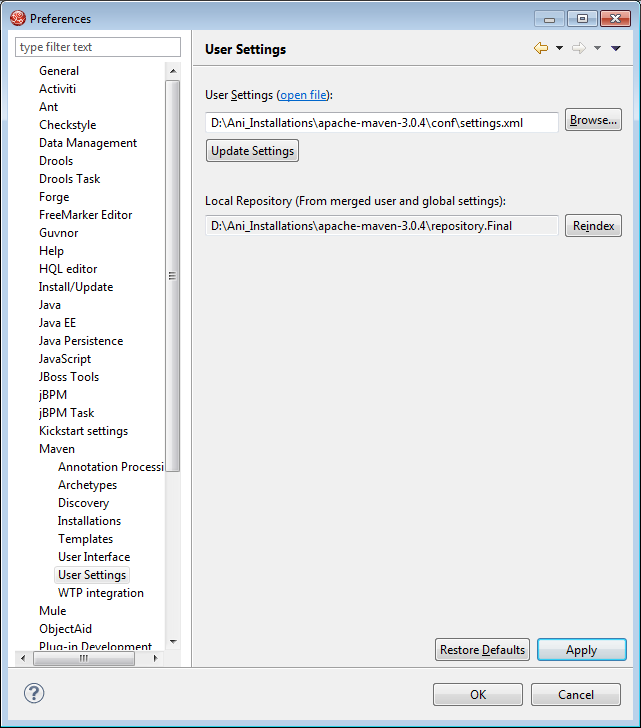
</proxy>

### Configuration

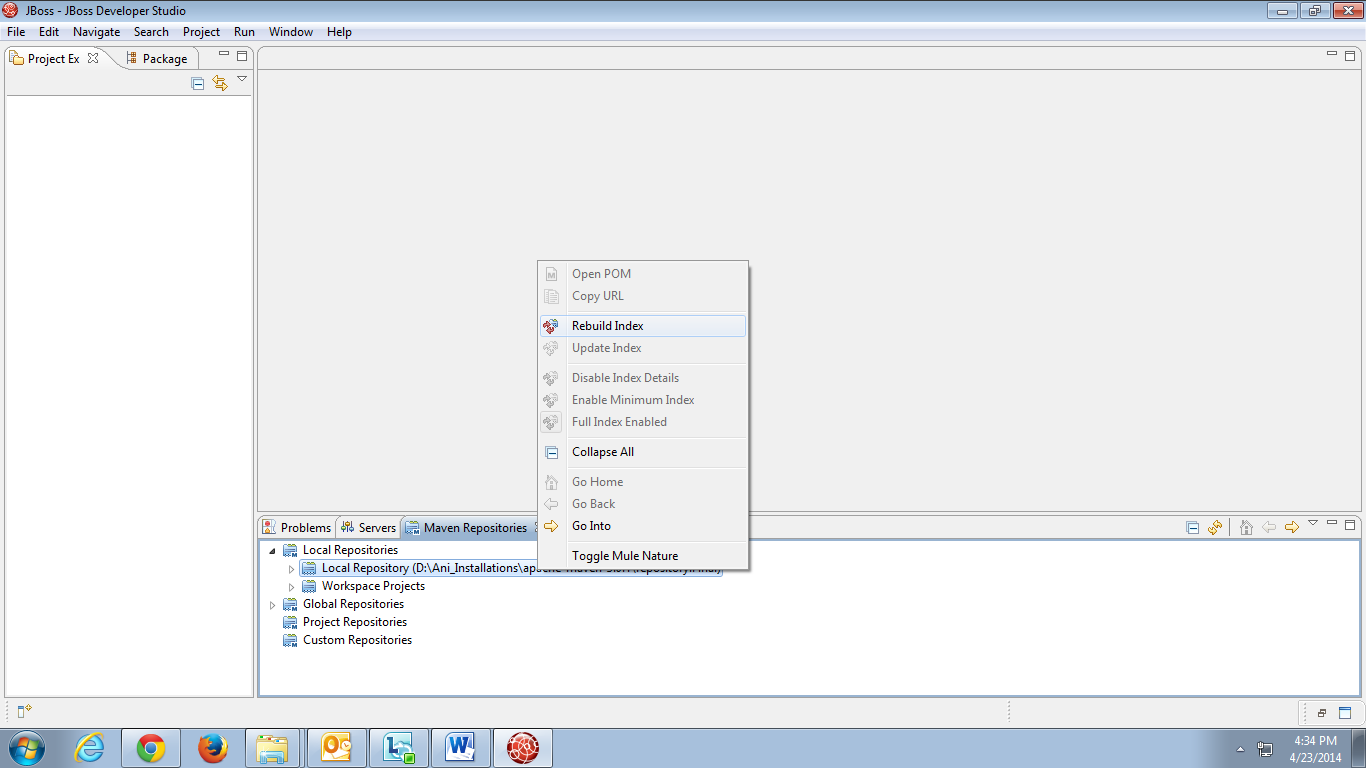
1. Open Window-Preferences-Maven in IDE.
2. Expand Maven.
3. Click on Installations.



1. Add the maven installation from your local system.
2. Click on User Settings.



1. Browse and choose the settings.xml from conf folder inside MAVEN\_HOME.
2. Right Click and Rebuild Index to get the local repository reflected in IDE.



**Note: The highlighted ones need to be updated with respect to your system.**

## Version Control System

### Prerequisites

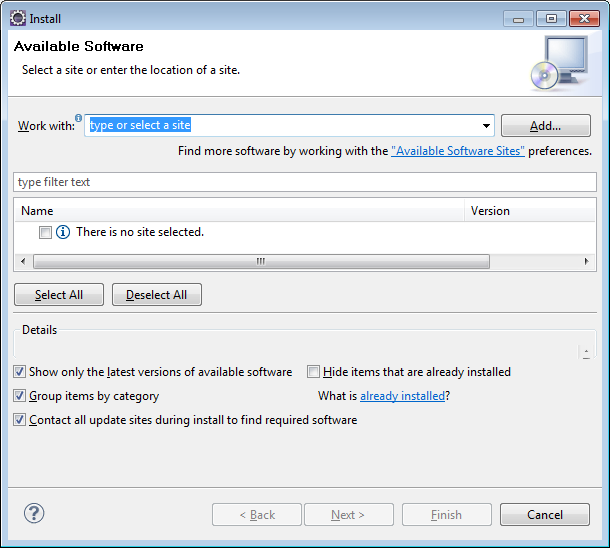
1. Make sure you have access to server 10.105.188.30 and SVN repositories. If not, refer the section [Getting Access](#_Getting_Access_1).
2. Make sure eclipse-SVN-site-1.8.5.zip exists in share folder 10.105.188.30/Softwares

### Installation

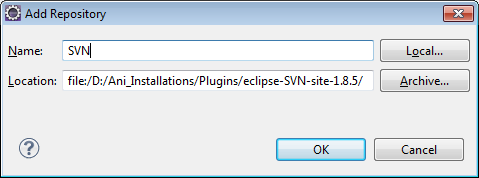
1. Copy the eclipse-SVN-site-1.8.5.zip file from the share folder.
2. Extract it to a folder in local drive.

### Configuration

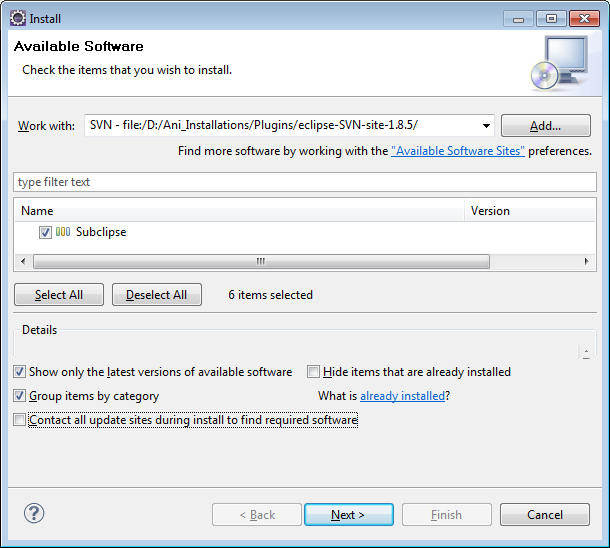
1. Open ‘Install New Software’ from Help-Install New Software in IDE.
2. Click Add.



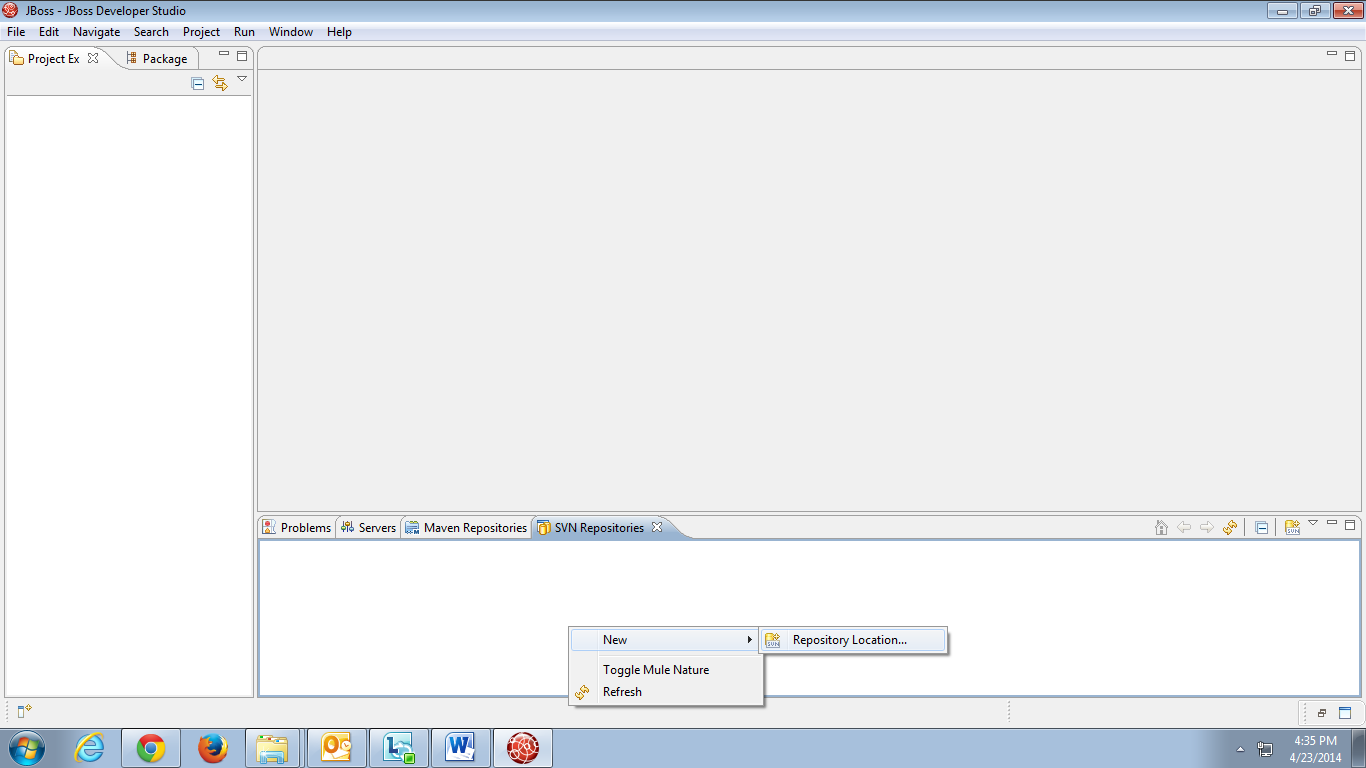
1. Choose the extracted folder and click OK.



1. The features will get listed in the screen.
2. Unselect ‘Contact all update sites during install to find required software’ check box.
3. Click Next.



1. Accept the agreement.
2. Once installed, restart the IDE if prompted.
3. Click on New- Repository Location.



Specify the URL and click Finish.

## Web Service Framework

### Prerequisites

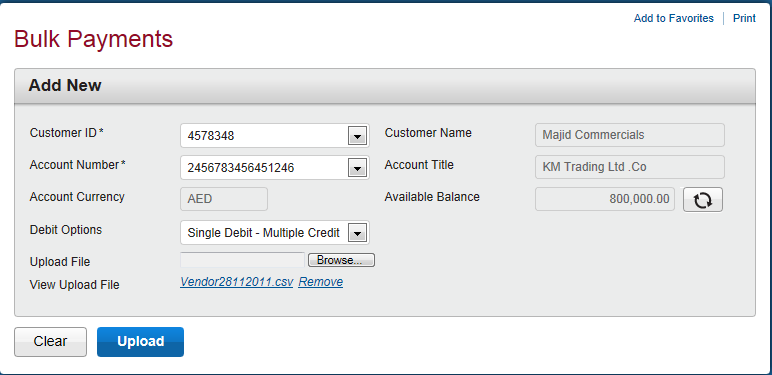
1. Make sure you have access to server 10.105.188.30. If not, refer the section [Getting Access](#_Getting_Access_1).
2. Make sure apache-cxf-2.5.2.zip exists in share folder 10.105.188.30/Softwares

### Installation

1. Copy the apache-cxf-2.5.2.zip file from the share folder.
2. Extract it to a folder in local drive.

# Business Layer Implementation

Below is the sample screen for which the services are identified, developed and explained in this guide.



Follow the instructions described in the below sections to get familiarize with the business layer implementation of First Connect.



## Prerequisites

### Identify Services

The first step is to identify the services to be exposed. For the above screen, the below services are required.

1. Service to fetch customer and account details.
2. Service to fetch account details.
3. Service to fetch debit options.
4. Service to accept the uploaded file along with other details.

### Identify API

For the identified services, identify the methods needs to be implemented and the source from where the details have to be fetched.

1. API to fetch customer ID and name from database.
2. API to fetch account number with respect to the customer ID from database.
3. API to fetch account balance, currency and title from T24.
4. API to fetch debit options from database.
5. API to store the details along with the file.

### Identify Request-Response

The input and output parameters needs to be identified.

1. Customer/Account details Service – Need to fetch customer ids and account numbers for which the logged in user is having access to.

**Request** – User Id

**Response** – List of customer Object with customer Id, name as fields and List of account numbers.

1. Account details Service – Need to fetch the account title, currency and balance from T24.

**Request** – Account Number

**Response** – List of Account object with account details.

1. Debit options Service – Need to fetch the available debit option from database.

**Request** – NA

**Response** – List of debit options

1. Upload file Service – Need to accept the user selected details along with the file uploaded and provide a reference number as response.

**Request** - File, customer id, account number, debit option.

**Response**- Reference number

### Identify Projects

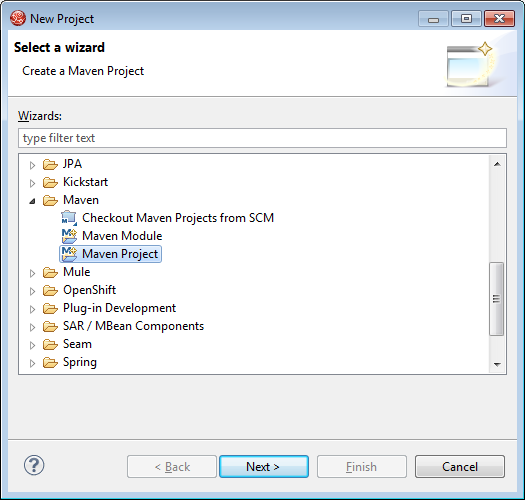
To achieve the above, the below projects are required.

1. BulkService – Web Project consists of web services and to be packed as war.
2. BulkPayment – Java Project consists of business logic and to be packed as jar.
3. BulkModel – Java Project consists of request and responses for the web service and to be packed as jar.
4. BulkDao – Java Project consists of methods interacting with database and to be packed as jar.

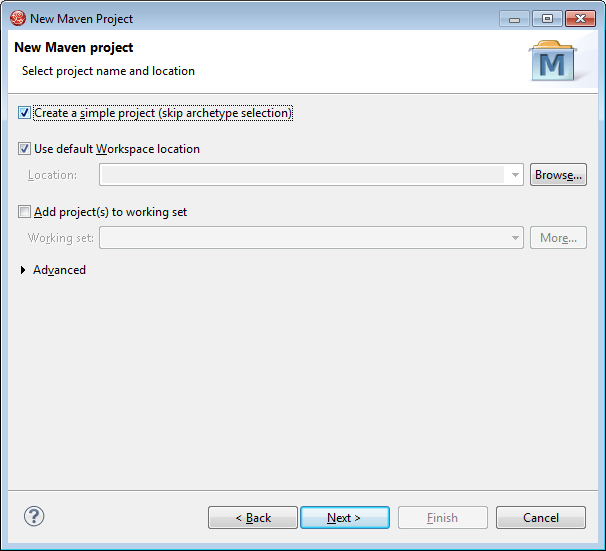
Note: If a project already exists, no need to create a new one, append your module specific packages to the project.

### [Web Project Creation](#_Project_Creation)

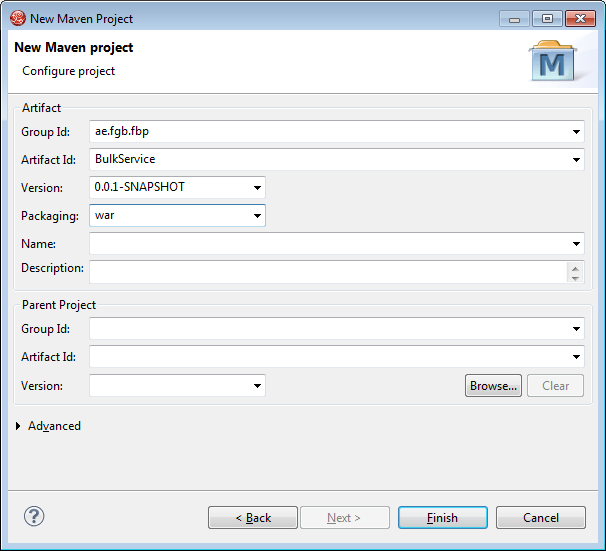
1. Click New – Maven Project.



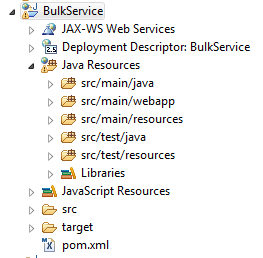
1. Select ‘Create a simple project’.



1. Enter the Group Id, Artifact Id and select Packaging as war.

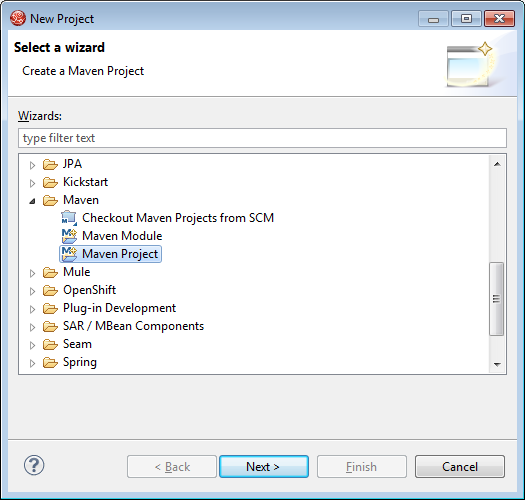


1. Click Finish and a Web Project is created with below structure.

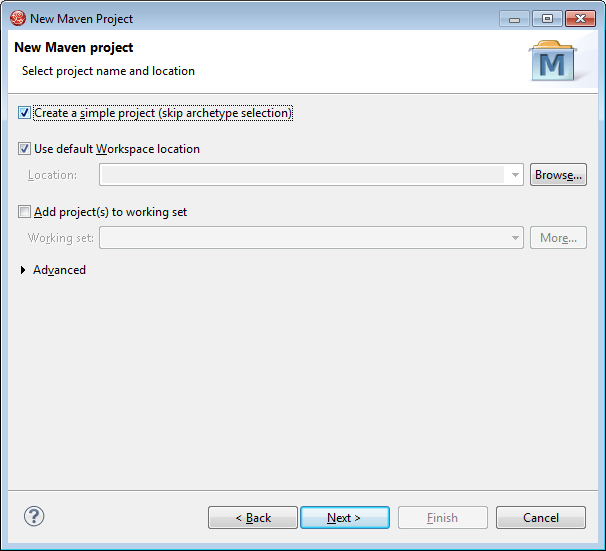


### [Java Project Creation](#_Web_Project_Creation)

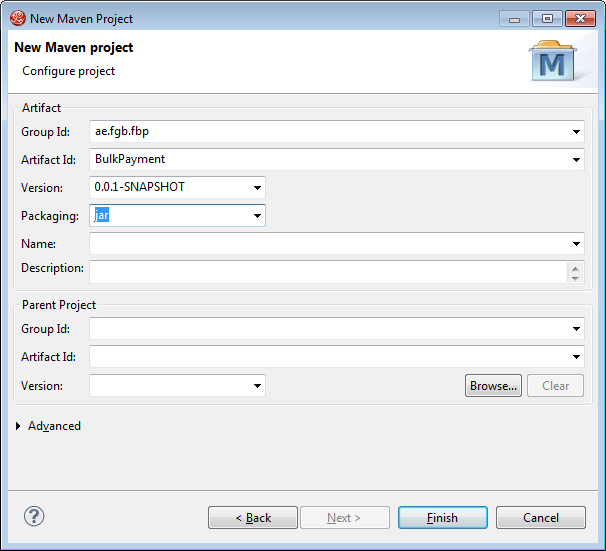
1. Click New – Maven Project.



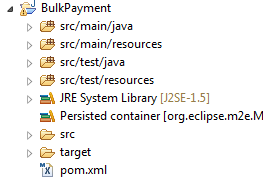
1. Select ‘Create a simple project’.



1. Enter the Group Id, Artifact Id and select Packaging as jar.



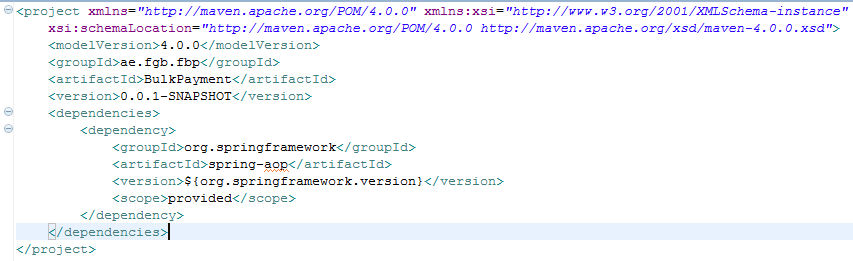
1. Click Finish and a Java Project is created with below structure.



Similarly create java projects for BulkModel and BulkDao.

### POM Creation

On creating a project, an xml with name pom.xml is created. This is used for managing dependencies automatically using Maven.



All the other project dependencies can also be added to this file and hence managed.

The web service project (BulkService) depends on business logic (BulkPayment), model (BulkModel) and frameworks.

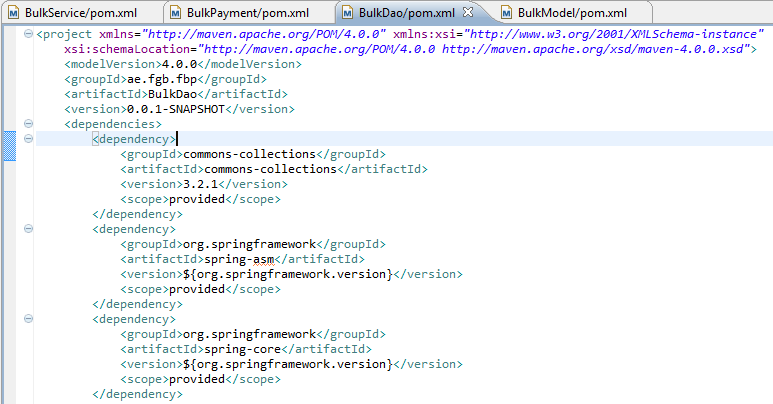


The business logic (BulkPayment) project depends on model, dao (BulkDao) and framework.



Model depends on Framework.

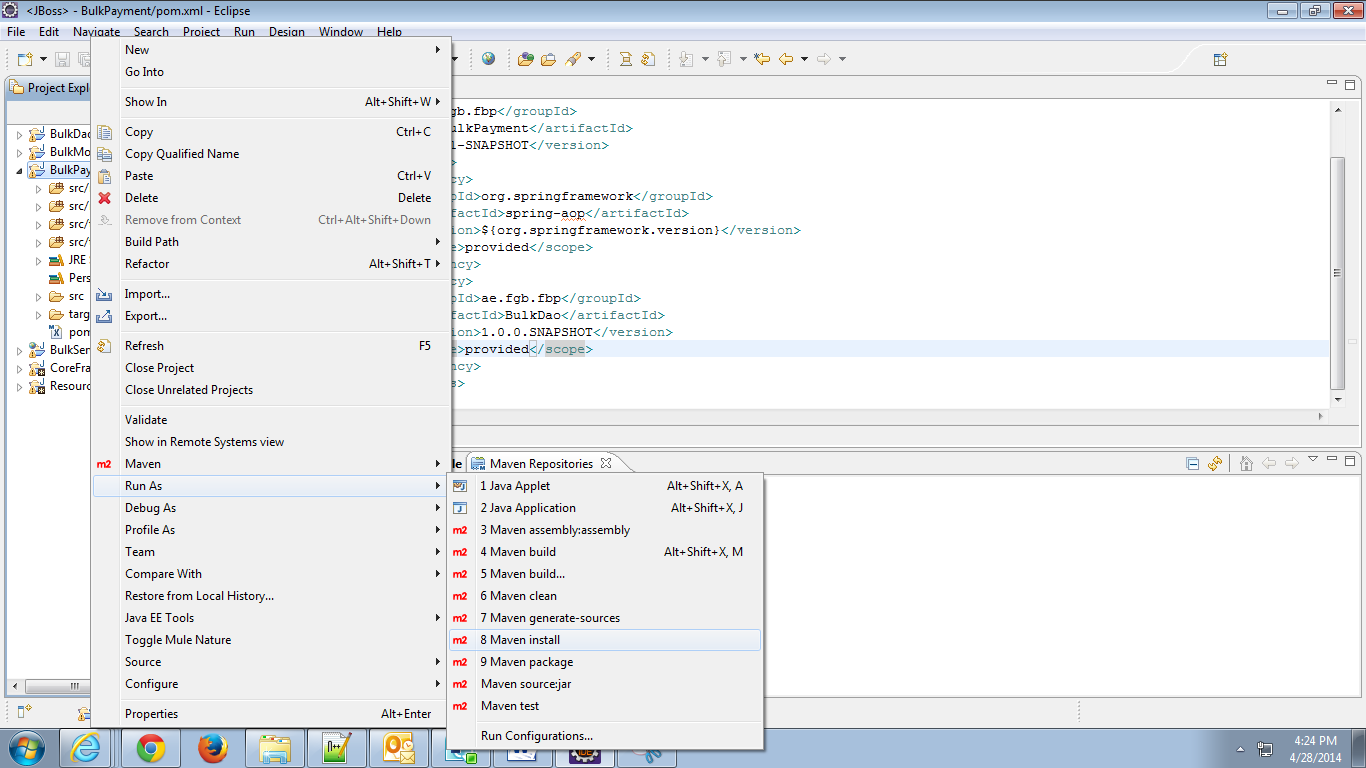




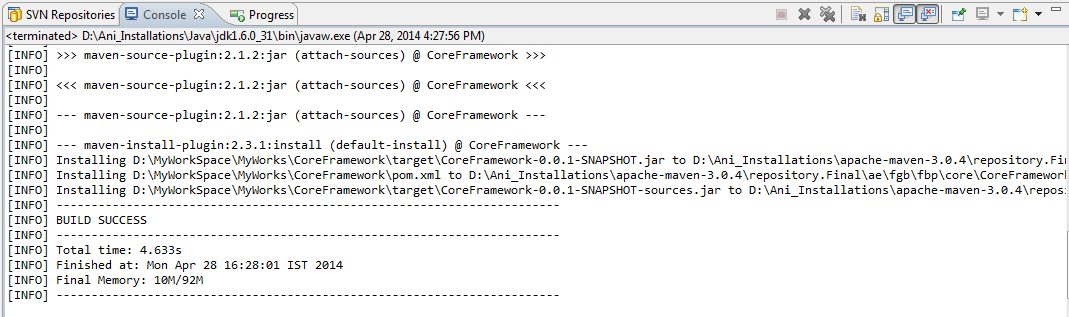
### Building Project

The Maven Project can be built to obtain either the jar or war as per the configuration. This will ensure that all the dependencies are also added.

For building the project, right click-run as – Maven install.



On successful build, it will show as ‘Build Success’.



If any errors occurred, refer Trouble Shooter.

## Exposing Web Service

### Files Required

For exposing web services, the following files are required.

1. End Point – Interface containing the service method signatures
2. End point implementation – Class implementing the interface.
3. Web.xml – xml having the servlet configurations.
4. ApplicationContext.xml - xml having the bean configurations.

### Request-Response Creation

Refer the section Identify Request-Response for details.

1. Customer/Account details Service and Debit options Service.

|  |
| --- |
| **public** **class** GenericRequest **extends** BaseServiceDO {  **private** String userId; |

|  |
| --- |
| **public** **class** GenericResponse **extends** BaseServiceDO {  **private** List<FbpCust> custs;  **private** List<FbpAcct> accts;  **private** List<FbpParamVal>debitOptions; |

1. Account Detail Service

|  |
| --- |
| **public** **class** AccountDetailsRequest **extends** BaseServiceDO {  **private** List<FbpAcct> accts = **null**;  **private** String chnl;  **private** **boolean** ownAcc; |

|  |
| --- |
| **public** **class** AccountDetailsResponse **extends** BaseServiceDO {  **private** List<FbpAcct> accts = **null**;  **private** List<FbpExtlAcct> extlAccts = **null**;  **private** List<FbpPymtTxnValidation> validations=**null**;  **private** **boolean** postRstct; |

1. Upload file Service

|  |
| --- |
| **public** **class** UploadRequest **extends** BaseServiceDO {  **private** FbpPymtReq pymtReq; |

|  |
| --- |
| **public** **class** UploadResponse **extends** BaseServiceDO {  **private** String refNmber; |

### End Point Creation

1. Create an interface with signatures of the identified services.

|  |
| --- |
| **public** **interface** BulkEndPoint {  GenericResponse getCustomerDetails(GenericRequest request);  AccountDetailsResponse getAccountDetails(AccountDetailsRequest request);  UploadResponse uploadFile(UploadRequest request);  GenericResponse getDebitOptions(GenericRequest request);  } |

1. Annotate the interface as web service.

|  |
| --- |
| @WebService  **public** **interface** BulkEndPoint { |

1. Annotate the methods, parameters and result.

|  |
| --- |
| @WebMethod  @WebResult(name = "CustResponse")  GenericResponse getCustomerDetails(  @WebParam(name = "CustRequest") GenericRequest request); |

1. Final end point will look as below.

|  |
| --- |
| @WebService  **public** **interface** BulkEndPoint {  @WebMethod  @WebResult(name = "CustResponse")  GenericResponse getCustomerDetails(  @WebParam(name = "CustRequest") GenericRequest request);  @WebMethod  @WebResult(name = "AccResponse")  AccountDetailsResponse getAccountDetails(  @WebParam(name = "AccRequest") AccountDetailsRequest request);  @WebMethod  @WebResult(name = "FileResponse")  UploadResponse uploadFile(  @WebParam(name = "FileRequest") UploadRequest request);  @WebMethod  @WebResult(name = "debitResponse")  GenericResponse getDebitOptions(  @WebParam(name = "debitRequest") GenericRequest request);  } |

### End Point Implementation Creation

1. Create a class that implements the above mentioned interface.

|  |
| --- |
| **public** **class** BulkEndPointImpl **implements** BulkEndPoint{ |

1. Annotate the class with @WebService.

|  |
| --- |
| @WebService(endpointInterface="ae.fgb.fbp.endpoint.sl.bulk.BulkEndPoint")  **public** **class** BulkEndPointImpl **implements** BulkEndPoint{ |

1. Inject the business interface.

|  |
| --- |
| **private** **static** IBulkPayment *bulk*=**null**;  **public** **static** **void** setBulk(IBulkPayment bulk) {  BulkEndPointImpl.*bulk* = bulk;  } |

1. Invoke the methods.

|  |
| --- |
| **public** GenericResponse getCustomers(GenericRequest request)  **throws** Throwable {  **return** *bulk*.getCustomers(request);  }  **public** AccountDetailsResponse getAccountDetails(  AccountDetailsRequest request) **throws** Throwable {  **return** *bulk*.getAccountDetails(request);  }  **public** UploadResponse uploadFile(UploadRequest request) {  **return** *bulk*.uploadFile(request);  }  **public** GenericResponse getDebitOptions(GenericRequest request)  **throws** Throwable {  **return** *bulk*.getDebitOptions(request);  } |

### Servlet Configurations

1. Add the below in web.xml which is located in WEB-INF under src/main/webapp.

|  |
| --- |
| <servlet>  <servlet-name>Bulk</servlet-name>  <servletclass>ae.fgb.fbp.endpoint.sl.bulk.BulkEndPointImpl  </servlet-class>  </servlet>  <servlet-mapping>  <servlet-name>Bulk</servlet-name>  <url-pattern>/Bulk</url-pattern>  </servlet-mapping> |

1. Pass the application Context path as a context param and configure a contextLoaderListener.

|  |
| --- |
| <context-param>  <param-name>contextConfigLocation</param-name>  <param-value>classpath:applicationcontext.xml</param-value>  </context-param>  <listener>  <listener-class>org.springframework.web.context.ContextLoaderListener  </listener-class>  </listener> |

### Bean Configurations

1. Create an xml, applicationcontext.xml in src/main/resources.
2. The beans for endpoint, business logic are created in the context created in business logic project (BulkPayment). Here just import that xml.

|  |
| --- |
| <import resource=*"classpath:bulkAppContext.xml"* /> |

## Business Logic Implementation

### Files Required

Following files are required for business implementation.

1. Business Interface
2. Business logic
3. ApplicationContext.xml

### Business Interface Creation

1. Create an interface with the necessary API’s identified in section Identify API.

|  |
| --- |
| **public** **interface** IBulkPayment {  GenericResponse getCustomers(GenericRequest request) **throws** Throwable;  AccountDetailsResponse getAccountDetails(AccountDetailsRequest request)**throws** Throwable;  UploadResponse uploadFile(UploadRequest request);  GenericResponse getDebitOptions(GenericRequest request) **throws** Throwable;  } |

### Business Logic Implementation Creation

1. Create a class that implements the above said interface.

|  |
| --- |
| **public** **class** Bulk **implements** IBulkPayment{  **public** GenericResponse getCustomers(GenericRequest request)  **throws** Throwable {  **return** **null**;  }  **public** AccountDetailsResponse getAccountDetails(  AccountDetailsRequest request) **throws** Throwable {  **return** **null**;  }  **public** UploadResponse uploadFile(UploadRequest request) {  **return** **null**;  }  **public** GenericResponse getDebitOptions(GenericRequest request)  **throws** Throwable {  **return** **null**;  }  } |

1. Inject dao interface and delegate.

|  |
| --- |
| **public** **class** Bulk **implements** IBulkPayment {  **private** IBulkDao dao;  **private** PaymentDelegate delegate; |

1. Implement the actual logic in each method.

### Bean Configurations

1. Create a spring configuration xml with all necessary bean definitions.

|  |
| --- |
| <!-- EndPoint configuration starts -->  <bean id=*"bulkEndpoint"* class=*"ae.fgb.fbp.endpoint.sl.bulk.BulkEndPointImpl"*>  <property name=*"bulk"* ref=*"bulk"*></property>  </bean>  <!-- EndPoint configuration ends -->  <!-- Payment Functional implementation configuration starts -->  <bean id=*"bulk"* class=*"ae.fgb.fbp.payments.sl.bulk.BulkPayment"*></bean>  <!-- Payment Functional implementation configuration ends --> |

## DAO Implementation

This covers how to interact with database.

### Files Required

For implementing data access layer, following files are required.

1. DAO Interface
2. DAO Implementation
3. Transformers
4. ApplicationContext.xml

### DAO Interface Creation

Create an interface that extends GenericDao which already exists in DataAccess Project. Genericdao contains methods to save, delete, and update a single object as well as collection of objects.

|  |
| --- |
| **public** **interface** IBulkDao **extends** IGenericDao<DomainObject> { |

Create necessary methods as identified in Identify API.

|  |
| --- |
| List<FbpCust> getCustomers(String userId) **throws** Throwable;  List<FbpAcct> getAccountNumbers(List<FbpCust> customers) **throws** Throwable;  List<FbpParamVal> getParamValues(FbpParam fbpParam)**throws** Throwable; |

Finally the interface will look like this.

|  |
| --- |
| **public** **interface** IBulkDao **extends** IGenericDao<DomainObject> {  List<FbpCust> getCustomers(String userId) **throws** Throwable;  List<FbpAcct> getAccountNumbers(List<FbpCust> customers) **throws** Throwable;  List<FbpParamVal> getParamValues(FbpParam fbpParam) **throws** Throwable;  } |

### DAO Implementation Creation

Create a class that implements the above created interface.

|  |
| --- |
| **public** **class** BulkDaoImpl **implements** IBulkDao {  **public** List<FbpCust> getCustomers(String userId) **throws** Throwable {  **return** **null**;  }  **public** List<FbpAcct> getAccNumbers(List<FbpCust> customers)**throws** Throwable {  **return** **null**;  }  **public** List<FbpParamVal> getParamValues(FbpParam fbpParam) **throws** Throwable{  **return** **null**;  }  } |

* Identify the POJO in which the query has to be executed.

Pass that POJO to Detached Criteria.

|  |
| --- |
| DetachedCriteria userGroup = DetachedCriteria.*forClass*(FbpCustUser.**class**); |

* Identify the list of fields required to be projected.

Set that in Projection List.

|  |
| --- |
| userGroup.setProjection(Projections.*property*("fbpCustGroup.custGroupId")); |

* Identify the conditions to be imposed on.

Set those in Restrictions.

|  |
| --- |
| userGroup.add(Restrictions.*eq*("userId", userId)); |

* Execute those criteria using hibernate Template, which is injected in GenericDao.

|  |
| --- |
| List<Object[]> objects = getHibernateTemplate().findByCriteria(cust); |

* The obtained results can be set into a POJO and returned back.

|  |
| --- |
| **for** (Object[] data : objects) {  FbpCust customer = **null**;  customer = **new** FbpCust();  customer.setCustId((String) data[0]);  customer.setCustFrstNm1((String) data[1]);  customer.setCustLastNm1((String) data[2]);  customers.add(customer);  } |

Finally, it will look like as below.

|  |
| --- |
| **public** List<FbpCust> getCustomers(String userId) **throws** Throwable {  List<FbpCust> customers = **new** ArrayList<FbpCust>();  DetachedCriteria userGroup = **null**;  DetachedCriteria custAssn = **null**;  DetachedCriteria cust = **null**;  userGroup = DetachedCriteria.*forClass*(FbpCustUser.**class**);  userGroup.setProjection(Projections.*property*("fbpCustGroup.custGroupId"));  userGroup.add(Restrictions.*eq*("userId", userId));  custAssn = DetachedCriteria.*forClass*(FbpCustGroupAssn.**class**);  custAssn.setProjection(Projections.*property*("id.custId"));  custAssn.add(Subqueries.*propertyEq*("id.custGroupId", userGroup));  cust = DetachedCriteria.*forClass*(FbpCust.**class**);  cust.add(Subqueries.*propertyIn*("custId", custAssn));  ProjectionList projList = Projections.*projectionList*();  projList.add(Projections.*property*("custId"));  projList.add(Projections.*property*("custFrstNm1"));  projList.add(Projections.*property*("custLastNm1"));  cust.setProjection(projList);  List<Object[]> objects = getHibernateTemplate().findByCriteria(cust);  **for** (Object[] data : objects) {  FbpCust customer = **null**;  customer = **new** FbpCust();  customer.setCustId((String) data[0]);  customer.setCustFrstNm1((String) data[1]);  customer.setCustLastNm1((String) data[2]);  customers.add(customer);  }  **return** customers;  } |

### Transformers Creation

In the above method, result POJO is getting formed in the same method. This alone can be moved to a separate class if the resultant projections are many.

The same example is restructured as below.

Before executing the criteria, add the transformer to the criteria as below.

|  |
| --- |
| cust.setResultTransformer(**new** CustomerInfoTransformer()); |

Create a transformer implementing ResultTransformer.

|  |
| --- |
| **public** **class** CustomerInfoTransformer **implements** ResultTransformer { |

Override the methods

|  |
| --- |
| **public** Object transformTuple(Object[] tuple, String[] aliases) {  **return** **null**;  } |

As below.

|  |
| --- |
| **public** FbpCust transformTuple(Object[] rowData, String[] aliases) {  FbpCust customer = **null**;  **if** (rowData != **null** && rowData.length >= 4) {  customer = **new** FbpCust();  FbpInst instance = **new** FbpInst();  FbpCmpny cmpny=**new** FbpCmpny();  instance.setInstId((String) rowData[1]);  customer.setCustId((String) rowData[0]);  customer.setFbpInst(instance);  customer.setCustFrstNm1((String) rowData[2]);  customer.setCustLastNm1((String) rowData[3]);  cmpny.setCmpnyCd((String) rowData[4]);  customer.setFbpCmpny(cmpny);  }  **return** customer;  } |

### Bean Creation

Create a spring configuration file with necessary bean definitions.

|  |
| --- |
| <bean id=*"bulkDao"* class=*"ae.fgb.fbp.dao.bulk.BulkDaoImpl"*>  <property name=*"sessionFactory"* ref=*"sessionFactory"*/>  </bean> |

The other bean definitions are already available in the dao-context.xml.

|  |
| --- |
| <bean id=*"dataSource"* class=*"org.springframework.jndi.JndiObjectFactoryBean"*>  <property name=*"jndiName"* value=*"OracleDS"* />  </bean>  <bean id=*"sessionFactory"*  class=*"org.springframework.orm.hibernate3.LocalSessionFactoryBean"*>  <property name=*"dataSource"* ref=*"dataSource"*></property>  <property name=*"configLocation"* value=*"classpath:hibernate.cfg.xml"/*> </bean> |

## ESB Integration

### Files Required

For fetching the account balance, there requires a T24 call made via ESB. Following are required to integrate ESB with business layer.

1. ESB service definition file – Developed and provided by ESB resource
2. Business Delegate
3. Bean configuration

The ESB service definition file looks like below.

|  |
| --- |
| @WebService  **public** **interface** ValidateAccNumService {  @WebResult(name = "valAccNumResponse")  ValAccNumResponse validateAccNumberDetails(  @WebParam(name = "valAccNumRequest") ValAccNumRequest valAccNumRequest); |

### Delegate Creation

Delegate is used to invoke ESB services.

1. Create a class with the ESB service injected in.

|  |
| --- |
| **public** **class** PaymentDelegate {  **private** ValidateAccNumService validate = **null**;  **public** **void** setValidate(ValidateAccNumService validate) {  **this**.validate = validate;  } |

1. Invoke the method.

|  |
| --- |
| **public** ESBPymtContext getAccountNumberDetails(ESBPymtContext request) **throws** Throwable {  ValAccNumRequest esbRequest = **null**;  ValAccNumResponse esbResponse = **null**;  esbRequest = prepareValidateRequest(request);  **try** {  esbResponse = validate.validateAccNumberDetails(esbRequest);  } **catch** (Throwable throwable) {  **throw** throwable;  }  **if** (esbResponse != **null**) {  request = prepareValidateResponse(esbResponse, request);  **return** request;  } **else** {  **throw** **new** NullPointerException();  }  } |

### Configurations

1. Add an entry in spring configuration file created in step Bean Configurations.

|  |
| --- |
| <!-- Delegate Configuration starts -->  <bean id=*"delegate"* class=*"ae.fgb.fbp.payments.sl.delegate.PaymentDelegate"*>  <property name="*validate*" ref="*validateClient*"></property>  </bean>  <!-- Delegate Configuration ends --> |

1. Create a factory for ESB service which is used to consume that service.

|  |
| --- |
| <!-- ESB factory Configuration starts -->  <bean id=*"validateClient"* factory-bean=*"validateFactory"*  factory-method=*"create"* />  <bean id=*"validateFactory"* class=*"org.apache.cxf.jaxws.JaxWsProxyFactoryBean"*>  <property name=*"serviceClass"*  value=*"ae.fgb.fbp.integration.endpoint.ValidateAccNumService"* />  <property name=*"address"* value=*"${esb.validation}"* />  </bean> |

Note: The highlight denotes the url in which the service is hosted.

## Framework Integration

All request and response should extend BaseServiceDO.

All request and response should carry service context and user context.

When returning response from a method, the service context from the request should be set in the response.

### Logging

The Core Framework will take care of logging, provided the below configuration is in place.

All the methods whose request and response extends BaseServiceDO can be logged.

|  |
| --- |
| <aop:config>  <aop:aspect ref=*"advice"*>  <aop:pointcut id=*"bulkPointcut"*  expression=*"execution(\* ae.fgb.fbp.bulk.sl.IBulk.\*(..))"* />  <aop:around pointcut-ref=*"bulkPointcut"* method=*"invoke"* />  </aop:aspect>  </aop:config> |

### Custom Logging

Apart from the above configuration, if any specific logging needs to be done, make use of the methods available in CoreLogger class in Core Framework project.

|  |
| --- |
| CoreLogger.*log*(*LOG*, CoreLogger.*INFO*,"Workflow manager-->authorizeTxn method has thrown Error :::",e); |

Or like below.

|  |
| --- |
| CoreLogger.*log*(**this**.getClass(),authorizeTxnRequest.getServiceContext(), CoreLogger.*INFO*,"authorizeTxn ", "Authorized: " + isSuccess, **null**); |

### Exception Handling

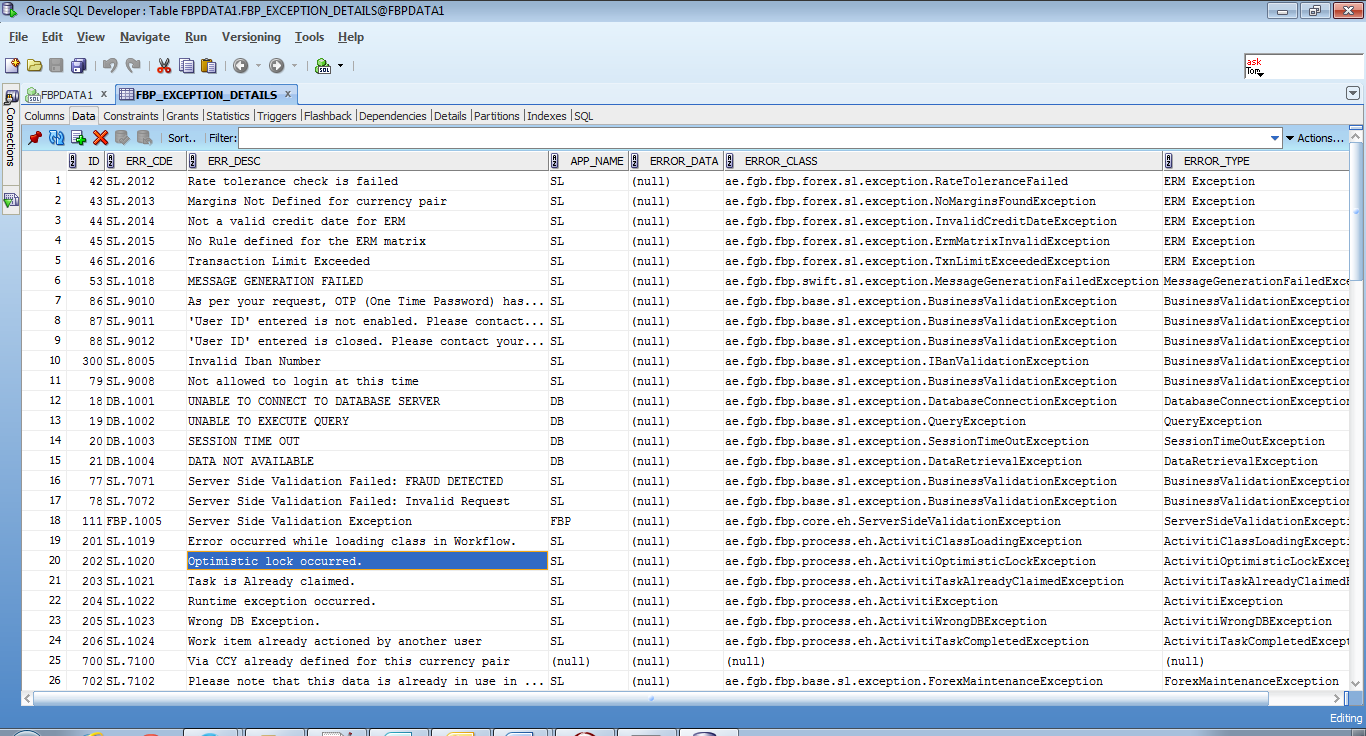
All the exceptions are handled and the custom- developer created exceptions are to be thrown.

All the exceptions are created in the project ‘GlobalException’.

1. Create exceptions in Global Exception under module specific packages extending GenericException which is available in Core Framework.
2. Create the constructors available from the super class.
3. Inject ExceptionType and provide a getter.

|  |
| --- |
| @WebFault  **public** **class** DataRetrievalException **extends** GenericException {  **private** **static** **final** **long** *serialVersionUID* = -1488632525177985351L;  **private** ExceptionType exceptionType;  **public** DataRetrievalException(ExceptionType exceptionType, Throwable cause) {  **super**(exceptionType, cause);  **this**.exceptionType = exceptionType;  }  **public** DataRetrievalException(ExceptionType exceptionType) {  **super**(exceptionType);  **this**.exceptionType = exceptionType;  }  **public** DataRetrievalException(String message, Throwable cause) {  **super**(message, cause);  }  **public** DataRetrievalException(String message) {  **super**(message);  }  **public** ExceptionType getExceptionType() {  **return** exceptionType;  }  } |

1. Place an entry for each exceptions created, in FBPExceptionDetail table.



1. Use the EhUtil class in GlobalException to consume the same in the code.
2. It already contains methods for DB related exceptions, ESB related exceptions, Business Validation and Data Unavailable exceptions.
3. Consume those like below.

If you want to throw data not available exception,

|  |
| --- |
| **throw** EhUtil.*getDataUnAvailableException*(); |

1. For the newly created exceptions, use the method createException() as below.

|  |
| --- |
| **throw** EhUtil.*createException*(SL,1000,e); |

The last parameter takes throwable. Pass it if available, else pass it as null.

### WS Security

Add a security interceptor in the bean that is created to consume the ESB services.

|  |
| --- |
| <property name=*"outInterceptors"*>  <list>  <ref bean=*"wsSecurityInterceptor"* />  </list>  </property> |

It will look like,

|  |
| --- |
| <bean id=*"validateClient"* factory-bean=*"validateFactory"*  factory-method=*"create"* />  <bean id=*"validateFactory"* class=*"org.apache.cxf.jaxws.JaxWsProxyFactoryBean"*>  <property name=*"serviceClass"*  value=*"ae.fgb.fbp.integration.endpoint.ValidateAccNumService"* />  <property name=*"address"* value=*"${esb.validation}"* />  <property name=*"outInterceptors"*>  <list>  <ref bean=*"wsSecurityInterceptor"* />  </list>  </property>  </bean> |

Refer Configurations for bean configuration without this security interceptor.

### Timeout

Add the below entry to the service, which may expect a time out.

|  |
| --- |
| <ref bean=*"timeoutSetter"* /> |

Finally, it will look this.

|  |
| --- |
| <bean id=*"swiftClient"* factory-bean=*"swiftFactory"* factory-method=*"create"* />  <bean id=*"swiftFactory"* class=*"org.apache.cxf.jaxws.JaxWsProxyFactoryBean"*>  <property name=*"serviceClass"*  value=*"ae.fgb.fbp.integration.endpoint.SendSwiftMsgService"* />  <property name=*"address"* value=*"${esb.swift}"* />  <property name=*"outInterceptors"*>  <list>  <ref bean=*"wsSecurityInterceptor"* />  <ref bean=*"timeoutSetter"* />  </list>  </property>  </bean> |

## Deployment

1. Build all the required projects.
2. Move the war file to the app server.
3. Start the server.
4. Hit the WSDL in browser to ensure everything is deployed correctly.

## Unit Testing

1. Create a java project.
2. Create a spring configuration file with client factory for the business service as specified in Bean configuration and necessary beans, if required.

|  |
| --- |
| <bean id=*"bulk"* factory-bean=*"pymtFactory"* factory-method=*"create"*></bean>  <bean id=*"pymtFactory"* class=*"org.apache.cxf.jaxws.JaxWsProxyFactoryBean"*>  <property name=*"serviceClass"*  value=*"ae.fgb.fbp.endpoint.sl.bulk.BulkEndPoint"* />  <property name=*"address"*  value=*"http://10.105.67.48:8180/Test/Test"* />  </bean> |

1. Create request object and invoke the service.

|  |
| --- |
| **public** **class** SLTest {    **static** ClassPathXmlApplicationContext *context*=**new** ClassPathXmlApplicationContext("client-beans.xml");  **static** ServiceContext *serviceContext*=**new** ServiceContext();  **static** UserContext *userContext*=**new** UserContext();  **static** BulkEndPoint *pymtEP*=(BulkEndPoint) *context*.getBean("pymt");  **public** **static** **void** main(String[] args) {  *getAccDetails*();  }    **public** **static** **void** getAccDetails(){  GenericResponse response;  GenericRequest request=**new** GenericRequest();  request.setUserId("FGBUSER1");  accDetRequest.setServiceContext(*serviceContext*);  **try**{  response=pymtEP.getAccountDetails(request);  }**catch**(){  e.printStackTrace();  }  }  } |

1. Similarly test the DAO methods individually.

## Existing Project Structure

* FCService – Web Project to which the services can be added without creating a new project.
* DataAccess – Java Project for having interactions with Database. Add interfaces/implementations to this project.
* Payments – Java Project for having implementations related to Payments.
* Forex
* UserAdmin
* Account Services
* Reports
* CommonsBL
* GlobalException
* CoreFramework
* Resource Bundle
* Security
* Generics
* Alerts
* Model – Java Project containing all domain model objects. Create request and responses in this project under appropriate packages.

# Trouble Shooter

## Build Management

1. **On building a project, if an exception of ‘dependency cannot be resolved’ comes**

* Check the local repository for the jar file.
* If found in system, rebuild the repository index in eclipse.
* If not found, check the proxy and build the project again. The jar should get downloaded.
* If not downloaded, check the internet settings in eclipse that it points to proxy.
* If not, download manually and place it in repository.

## Application Servers

1. **Address already bind exception**

* Check whether the server is already running. If yes, stop it.
* Check if any other services are running in that port. If yes, change the server’s port and then restart it.
* If no services are running but still the exception comes, execute the below command in command prompt.

netstat -n -a –o

* It will list out all the services running. Find out the PID of the service running in the port specified. Kill it as below.

taskkill /pid <PID>

Note: Kill the process only if you are sure that it is not required.

1. **Perm gen space**

* Increase the xmx and xms values in server configuration.

1. **Error opening in zip file**

* Stop the server.
* Remove the deployed war/ear.
* Deploy it again.
* Restart the server.

1. **Certificate Exception**

* Check whether the required certificate is present.
* Check whether the certificate is not expired.
* Check whether the certificate location is pointed properly.

## Deployment

1. **Marshalling exception**

* Check whether the same endpoints, requests, response are used in both server and client.

1. **Could not send message**

* Check whether the server is up and running.
* Check whether url pointing to is correct.
* Check whether the server is reachable from your side.
* Check whether any firewall is blocking.

1. **Could not resolve a binding when exposing web service**

* Check whether the cxf jars are available in jboss lib.