

*FOR:*

**Headstrong**

*CONTENT:*

Document for New Developer

*Version 1.1*

*April,15 2013*

L

**TABLE OF CONTENTS**

[1 introduction 3](#_Toc354064147)

[1.1 Purpose 3](#_Toc354064149)

[2 Audience 4](#_Toc354064150)

[3 Scope 4](#_Toc354064151)

[4 Setting up system with Teevra. 4](#_Toc354064152)

[4.1 Need to be installed: 4](#_Toc354064153)

[4.2 Checking out code from SVN. 5](#_Toc354064154)

[4.3 Build code 5](#_Toc354064156)

[4.4 Create new user for Database with all privileges. 8](#_Toc354064157)

[4.5 Encryption for password 9](#_Toc354064158)

[4.6 For DB Setup: 9](#_Toc354064159)

[4.7 Run start-Teevra.bat 9](#_Toc354064160)

[4.8 UI 10](#_Toc354064165)

[4.9 Run installer of Teevra. 11](#_Toc354064169)

[5 Processes in Teevra UI 11](#_Toc354064170)

[6 Component Development 11](#_Toc354064172)

[7 Event-driven endpoint implementation 11](#_Toc354064174)

[8 Writing service 13](#_Toc354064178)

1. introduction

Teevra is made of following layers:

1. UI Layer – This is a Rich Internet Application that runs on Flash player inside a web browser. This requires Flash Player 9 or above.
2. Services and Persistence Layer – This is the backend web component for our UI Layer and it runs within a Java Servlet container (web server).
3. Teevra Fusion Framework – Also called as Teevra Server. This is the backend server having all the required components for message flow and processing.
4. Teevra DB – Any JDBC compliant database shared by both Teevra Web application and Teevra server (Fusion Framework)
   1. Purpose

This document explains setting up Teevra project on once system and an overview of how it works and how to add new functionalities to it.

1. Audience

* Headstrong Product Manager
* Headstrong Product Development Team

1. Scope

This document describes what is there in Teevra, how it got built.

1. Setting up system with Teevra.
   1. Need to be installed:

### Hardware

* 4GB RAM
* Processor: 2.00 GHz or more
* 5Gb Disk space
* 1 CPU core
* Support for 32 bit system

### Software

* Windows 7 / Windows 2003 – 32 bit OS
* Internet Explorer 9
* Apache Tomcat 5.5 or above
* Oracle Java6 JRE (32 bit)
* Oracle (9i / 10g)
* Flash 9.0 or above
* Silverlight 4.0 or above
* Microsoft Visual Studio
* PuTTY
* TortoiseSVN
* apache-maven-2.0.9 – required to build the Teevra Server side code
* apache-ant-1.6.2 – Required to build the Teevra UI
* Eclipse Galileo

1. **For the Maven setup:**

1) Unpack the archive file apache-maven-2.0.x-bin.zip obtained from the following link: <http://maven.apache.org/download.html#Installation>

2) A directory called " apache-maven-2.0.x-bin " will be created.

                     3) Add the bin directory to your PATH, eg:

                For Windows 2000/XP:

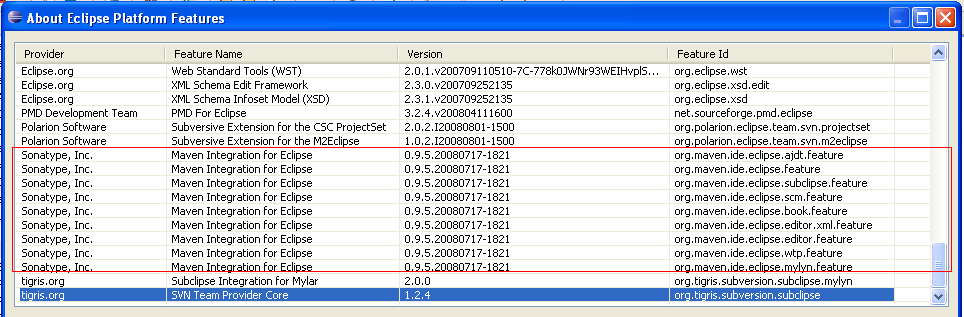
set PATH="c:\program files\apache-maven-2.0.x\bin";%PATH%

        4) JAVA\_HOME needs to be set to the location of your JDK

       5) Run "mvn --version" on console to verify that it is correctly installed.

1. **For installing Maven plugin into Eclipse –**
2. <http://www.roseindia.net/maven2/maven_2_eclipse_plugin.shtml>

(2)   We are currently using Eclipse-Europa (Eclipse Platform Version: 3.3.1.1) that perhaps is Maven Integrated by default. The plugin features of my Eclipse Platform shows the following:



* 1. Checking out code from SVN.

To checkout code from SVN.

One should have SVN repository link to that code.

Create any new folder and name it as Teevra or anything suitable, then go to SVN checkout, paste that SVN link and then press OK. It will check out all the code in that folder.

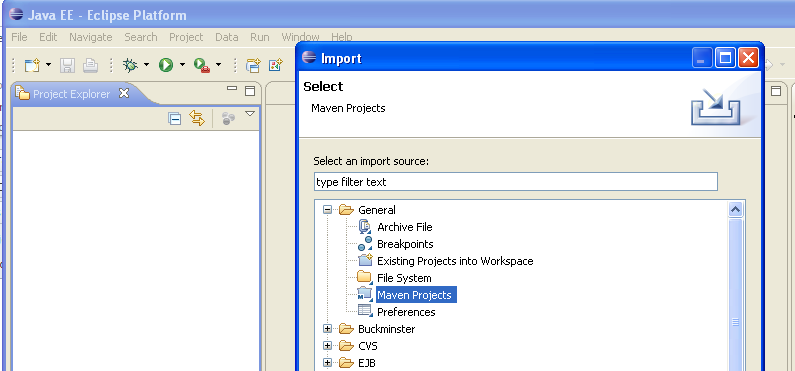
* 1. Build code

from ${CODEBASE}/TeevraServer/assembler/scripts/build-noTests.bat and run this script in command prompt. It should come as build successful.

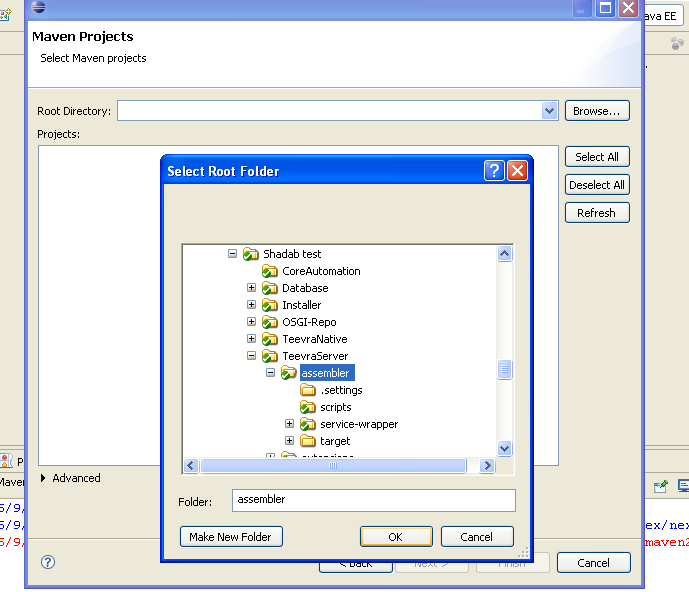
Now, import code in eclipse by selecting Maven project.

Below is another snapshot of how to import Maven projects into the eclipse (once the plugin is in place).

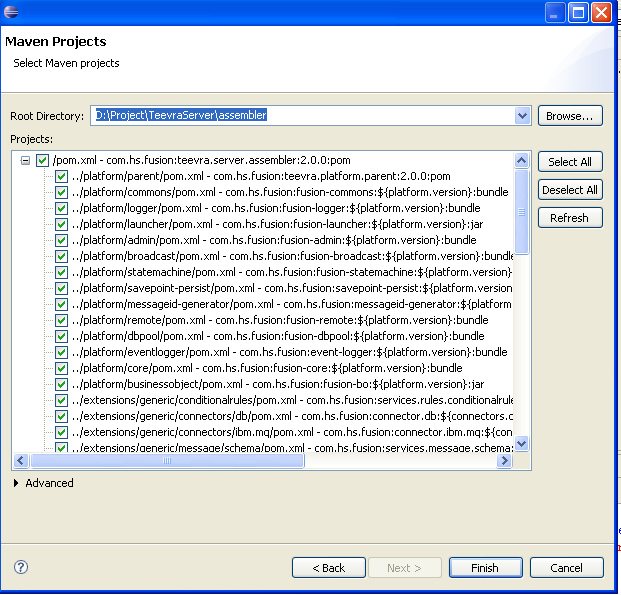
**Snap#1**



**Snap#2**



**Snap#3**



* 1. Create new user for Database with all privileges.
  2. Encryption for password

Copy local.policy.jar and US\_export\_policy.jar in${JAVA\_HOME}/*jdk1.6/jre/lib/security/*

Take key generation and encryption jars

*PassKeyGenerator.jar*

*PasswordEncrypter.jar*

And create two txt files- one for key and two for passwords (database password file and JMX password file).

Run first PassKeyGenerator.jar in command prompt and specify the key file path.

Then run PasswordEncrypter.jar in command prompt and specify the password file path where you created txt file for password with password file name *D:/password/pwd.txt (for your db)* and *D:/password/jmx.txt (*for jmx password which is teevra!123).

* 1. For DB Setup:

1. Import and run the script file: ‘*teevra\_ddl’* present in the following folder: **${CODEBASE}/***Database/PostgresScripts*. This sets up the table structure.
2. Run the following class: ‘DataLoader.java’ present in - **${CODEBASE}/***utilities/dataLoader/src/com/hs/stridehub/config/*loader which imports data from a ‘Tables.xls’ file into these tables.

The above two steps would be tedious. Hence the alternative approach is to simply do the following.

Run database scripts for new user created in oracle. For this, run the batch file from

**${CODEBASE}/**Installer/db/Launchsql.bat

but before running, edit this batch file with the paths of the jars and database details.

After running batch file, it will create all the tables in database and components in UI.

* 1. Run start-Teevra.bat

Edit this property file with JMX properties, database properties and encryption key path given in section 4.4

${CODEBASE}/*TeevraServer/assembler/target/teevra.server.assembler-2.0.0-bin.dir/conf/fusion.properties.*

fusion.properties

Configure the following:

#**Fusion server jmx properties**

fusion.jmx.url= service:jmx:rmi:///jndi/rmi://localhost:8004/jmxrmi

fusion.jmx.port=8004-----> (this could be any unused port. The same has to be configured in tomcat – described later)

fusion.jmx.username=teevra

fusion.jmx.password=

fusion.jmx.passwordLocation=D:/password/jmx.txt **----->( as per your password path generated in section 4.5)**

**#Teevra core db DB Properties**

database.driver.classname=oracle.jdbc.OracleDriver

database.username=teevra

database.password=

database.passwordLocation=D:/passwrd/tiger.txt **----->( as per your password path generated in section 4.5)**

database.url=jdbc:oracle:thin:@localhost:1521:xe **----->( as per your Db configuration)**

**#Encryption key path**

fusion.encryption.keyPath=D:/passwrd/key.txt ----->( as per your key encryption path)

* 1. UI
* Take *teevra.war* file
* Go to **${TOMCAT\_HOME}/***webapps/* folder
* Paste *teevra.war* file there.
* Start Tomcat server.
* Stop Tomcat server.
* Go to **${TOMCAT\_HOME}/***webapps/teevra/WEB-INF/classes/*properties folder
* Update *teevraApp.properties* file with jmx user and jmx password if it’s not there already.
* Outside this folder. i.e **${TOMCAT\_HOME}/***webapps/teevra/WEB-INF/*
* Update the following in *jdbc.properties* file with database properties and encryption key path.
* Edit *log4j.properties* file.
* Start the Tomcat Server.

jdbc.properties

Configure the following:

jdbc.driverClassName=oracle.jdbc.OracleDriver

jdbc.url=jdbc:oracle:thin:@localhost:1521:xe

jdbc.username=teevra

jdbc.password=

jdbc.passwordLocation=D:/passwrd/tiger.txt

hibernate.dialect=org.hibernate.dialect.OracleDialect

*Other alternative for section 4.5, 4.6 and 4.7 is to run installer.*

* 1. Run installer of Teevra.

For that, refer to installer guide.i.e. *Teevra\_InstallationGuide.docx*

When everything set up on your system, then open this link <http://localhost:8080/teevra>. It will show you the Teevra UI.

1. Processes in Teevra UI

Refer to *Teevra\_ScreenShots.docx*

For knowing about components in Teevra UI, Refer to *TeevraGateways\_Design\_FusionServer .doc*

1. Component Development

Refer to *Teevra -ServersideComponentDevGuide.docx*

1. Event-driven endpoint implementation

If your custom endpoint conforms to the event-driven pattern

it is implemented by extending the abstract class, *org.apache.camel.impl.DefaultEndpoint*, as shown in Example.

Example Implementing DefaultEndpoint:

**import java.util.Map;**

**import java.util.concurrent.BlockingQueue;**

**import org.apache.camel.Component;**

**import org.apache.camel.Consumer;**

**import org.apache.camel.Exchange;**

**import org.apache.camel.Processor;**

**import org.apache.camel.Producer;**

**import org.apache.camel.impl.DefaultEndpoint;**

**import org.apache.camel.impl.DefaultExchange;**

**public class *CustomEndpoint* extends DefaultEndpoint {**

**[1](http://fusesource.com/docs/router/2.4/prog_guide/Component-Impl-Endpoint.html#Component-IDE-Map01)**

**public *CustomEndpoint*(String endpointUri, Component component) { [2](http://fusesource.com/docs/router/2.4/prog_guide/Component-Impl-Endpoint.html#Component-IDE-Map02)**

**super(endpointUri, component);**

**// Do any other initialization...**

**}**

**public Producer createProducer() throws Exception { [3](http://fusesource.com/docs/router/2.4/prog_guide/Component-Impl-Endpoint.html#Component-IDE-Map03)**

**return new *CustomProducer*(this);**

**}**

**public Consumer createConsumer(Processor processor) throws Exception { [4](http://fusesource.com/docs/router/2.4/prog_guide/Component-Impl-Endpoint.html#Component-IDE-Map04)**

**return new *CustomConsumer*(this, processor);**

**}**

**public boolean isSingleton() {**

**return true;**

**}**

**// Implement the following methods, only if you need to set exchange properties.**

**//**

**public Exchange createExchange() { [5](http://fusesource.com/docs/router/2.4/prog_guide/Component-Impl-Endpoint.html#Component-IDE-Map05)**

**return this.createExchange(getExchangePattern());**

**}**

**public Exchange createExchange(ExchangePattern pattern) {**

**Exchange result = new DefaultExchange(getCamelContext(), pattern);**

**// Set exchange properties**

**...**

**return result;**

**}**

**}**

|  |  |
| --- | --- |
| [1](http://fusesource.com/docs/router/2.4/prog_guide/Component-Impl-Endpoint.html#Component-IDE-Co01) | Implement an event-driven custom endpoint, *CustomEndpoint*, by extending the *DefaultEndpoint* class. |
| [2](http://fusesource.com/docs/router/2.4/prog_guide/Component-Impl-Endpoint.html#Component-IDE-Co02) | You must have at least one constructor that takes the endpoint URI, *endpointUri*, and the parent *component* reference, component, as arguments. |
| [3](http://fusesource.com/docs/router/2.4/prog_guide/Component-Impl-Endpoint.html#Component-IDE-Co03) | Implement the *createProducer*() factory method to create producer endpoints. |
| [4](http://fusesource.com/docs/router/2.4/prog_guide/Component-Impl-Endpoint.html#Component-IDE-Co04) | Implement the *createConsumer()* factory method to create event-driven consumer instances.   |  |  | | --- | --- | | [Important] | Important | | Do *not* override the *createPollingConsumer()* method. | |
| [5](http://fusesource.com/docs/router/2.4/prog_guide/Component-Impl-Endpoint.html#Component-IDE-Co05) | In general, it is *not* necessary to override the *createExchange*() methods. The implementations inherited from *DefaultEndpoint* create a *DefaultExchange* object by default, which can be used in any Apache Camel component. If you need to initialize some exchange properties in the *DefaultExchange* object, however, it is appropriate to override the *createExchange*() methods here in order to add the exchange property settings. |

The *DefaultEndpoint* class provides default implementations of the following methods, which you might find useful when writing your custom endpoint code:

* *getEndpointUri()—Returns the endpoint URI.*
* *getCamelContext()—Returns a reference to the CamelContext.*
* *getComponent()—Returns a reference to the parent component.*
* *createPollingConsumer()—Creates a polling consumer. The created polling consumer's functionality is based on the event-driven consumer. If you override the event-driven consumer method, createConsumer(), you get a polling consumer implementation for free.*
* *createExchange(Exchange e)—Converts the given exchange object, e, to the type required for this endpoint. This method creates a new endpoint using the overridden createExchange() endpoints. This ensures that the method also works for custom exchange types.*

*Prior to this,we need to add components in factory.*

1. Writing service

For writing service, one needs to write *activator* for that service where we register the *service* and then in *service implementation* we can write process method with *MessageExchange* and *ServiceConfig* as arguments.

*public void process(MessageExchange exchange, ServiceConfig config)*

In this method , we can write its functionality.