Recueil des tutoriels utilisés

2010-2011

Dans ce document on retrouve tous les tutoriaux que nous avons lus et utilisés afin de mener à bien notre projet de développement d'une application de gestion d'emploi du temps des auditions

Document en anglais

Sommaire

1. HBase

Table of Contents

Requirements

- Java 1.6.x, preferably from Sun. Use the latest version available except u18 (u19 is fine).
- This version of HBase will only run on Hadoop 0.20.x.
- *ssh* must be installed and *sshd* must be running to use Hadoop's scripts to manage remote Hadoop daemons. You must be able to ssh to all nodes, including your local node, using passwordless login (Google "ssh passwordless login").
- HBase depends on ZooKeeper as of release 0.20.0. HBase keeps the location of its root table, who the current master is, and what regions are currently participating in the cluster in ZooKeeper. Clients and Servers now must know their *ZooKeeper Quorum locations* before they can do anything else (Usually they pick up this information from configuration supplied on their CLASSPATH). By default, HBase will manage a single ZooKeeper instance for you. In *standalone* and *pseudo-distributed* modes this is usually enough, but for *fully-distributed* mode you should configure a ZooKeeper quorum (more info below).
- Hosts must be able to resolve the fully-qualified domain name of the master.

- The clocks on cluster members should be in basic alignments. Some skew is tolerable but wild skew could generate odd behaviors. Run NTP on your cluster, or an equivalent.
- This is the current list of patches we recommend you apply to your running Hadoop cluster:
    - HDFS-630: *"In DFSOutputStream.nextBlockOutputStream(), the client can exclude specific datanodes when locating the next block"*. Dead DataNodes take ten minutes to timeout at NameNode. In the meantime the NameNode can still send DFSClients to the dead DataNode as host for a replicated block. DFSClient can get stuck on trying to get block from a dead node. This patch allows DFSClients pass NameNode lists of known dead DataNodes.
- HBase is a database, it uses a lot of files at the same time. The default **ulimit -n** of 1024 on *nix systems is insufficient. Any significant amount of loading will lead you to FAQ: Why do I see "java.io.IOException...(Too many open files)" in my logs?. You will also notice errors like:
- 2010-04-06 03:04:37,542 INFO org.apache.hadoop.hdfs.DFSClient: Exception increateBlockOutputStream java.io.EOFException
- 2010-04-06 03:04:37,542 INFO org.apache.hadoop.hdfs.DFSClient: Abandoning block blk_-6935524980745310745_1391901

Do yourself a favor and change this to more than 10k using the FAQ. Also, HDFS has an upper bound of files that it can serve at the same time, called xcievers (yes, this is *misspelled*). Again, before doing any loading, make sure you configured Hadoop's conf/hdfs-site.xml with this:

```
<property>
  <name>dfs.datanode.max.xcievers</name>
  <value>2047</value>
</property>
```

See the background of this issue here: Problem: "xceiverCount 258 exceeds the limit of concurrent xcievers 256". Failure to follow these instructions will result in **data loss**.

Windows

If you are running HBase on Windows, you must install Cygwin to have a *nix-like environment for the shell scripts. The full details are explained in the Windows Installation guide.

Getting Started

What follows presumes you have obtained a copy of HBase, see Releases, and are installing for the first time. If upgrading your HBase instance, see Upgrading.

Three modes are described: *standalone*, *pseudo-distributed* (where all servers are run on a single host), and *fully-distributed*. If new to HBase start by following the standalone instructions.

Begin by reading Requirements.

Whatever your mode, define ${HBASE_HOME} to be the location of the root of your HBase installation, e.g. /user/local/hbase. Edit ${HBASE_HOME}/conf/hbase-env.sh. In this file you can set the heapsize for HBase, etc. At a minimum, set JAVA_HOME to point at the root of your Java installation.

Standalone mode

If you are running a standalone operation, there should be nothing further to configure; proceed to Running and Confirming Your Installation. If you are running a distributed operation, continue reading.

Distributed Operation: Pseudo- and Fully-distributed modes

Distributed modes require an instance of the *Hadoop Distributed File System* (DFS). See the Hadoop requirements and instructions for how to set up a DFS.

Pseudo-distributed mode

A pseudo-distributed mode is simply a distributed mode run on a single host. Once you have confirmed your DFS setup, configuring HBase for use on one host requires modification of ${HBASE_HOME}/conf/hbase-site.xml, which needs to be pointed at the running Hadoop DFS instance. Use hbase-site.xml to override the properties defined in ${HBASE_HOME}/conf/hbase-default.xml (hbase-default.xml itself should never be modified). At a minimum the hbase.rootdir property should be redefined in hbase-site.xml to point HBase at the Hadoop filesystem to use. For example, adding the property below to your hbase-site.xml says that HBase should use the /hbase directory in the HDFS whose namenode is at port 9000 on your local machine:

```
<configuration>
  ...
  <property>
    <name>hbase.rootdir</name>
    <value>hdfs://localhost:9000/hbase</value>
    <description>The directory shared by region servers.
    </description>
  </property>
  ...
</configuration>
```

Note: Let HBase create the directory. If you don't, you'll get warning saying HBase needs a migration run because the directory is missing files expected by HBase (it'll create them if you let it).

Also Note: Above we bind to localhost. This means that a remote client cannot connect. Amend accordingly, if you want to connect from a remote location.

Fully-Distributed Operation

For running a fully-distributed operation on more than one host, the following configurations must be made *in addition* to those described in the pseudo-distributed operation section above.

In hbase-site.xml, set hbase.cluster.distributed to true.

```
<configuration>
  ...
  <property>
    <name>hbase.cluster.distributed</name>
    <value>true</value>
    <description>The mode the cluster will be in. Possible values are
      false: standalone and pseudo-distributed setups with managed Zookeeper
      true: fully-distributed with unmanaged Zookeeper Quorum (see hbase-env.sh)
    </description>
  </property>
  ...
</configuration>
```

In fully-distributed mode, you probably want to change your hbase.rootdir from localhost to the name of the node running the HDFS NameNode. In addition to hbase-site.xml changes, a fully-distributed mode requires that you modify ${HBASE_HOME}/conf/regionservers. The regionserver file lists all hosts running HRegionServers, one host per line (This file in HBase is like the Hadoop slaves file at ${HADOOP_HOME}/conf/slaves).

A distributed HBase depends on a running ZooKeeper cluster. All participating nodes and clients need to be able to get to the running ZooKeeper cluster. HBase by default manages a ZooKeeper cluster for you, or you can manage it on your own and point HBase to it. To toggle HBase management of ZooKeeper, use the HBASE_MANAGES_ZK variable in ${HBASE_HOME}/conf/hbase-env.sh. This variable, which defaults to true, tells HBase whether to start/stop the ZooKeeper quorum servers alongside the rest of the servers.

When HBase manages the ZooKeeper cluster, you can specify ZooKeeper configuration using its canonical zoo.cfg file (see below), or just specify ZookKeeper options directly in the ${HBASE_HOME}/conf/hbase-site.xml (If new to ZooKeeper, go the path of specifying your configuration in HBase's hbase-site.xml). Every ZooKeeper configuration option has a corresponding property in the HBase hbase-site.xml XML configuration file named hbase.zookeeper.property.OPTION. For example, the clientPort setting in ZooKeeper can be changed by setting the hbase.zookeeper.property.clientPort property. For the full list of available properties, see ZooKeeper's zoo.cfg. For the default values used by HBase, see ${HBASE_HOME}/conf/hbase-default.xml.

At minimum, you should set the list of servers that you want ZooKeeper to run on using the hbase.zookeeper.quorum property. This property defaults to localhost which is not suitable for a fully distributed HBase (it binds to the local machine only and remote clients will not be able to connect). It is recommended to run a ZooKeeper quorum of 3, 5 or 7 machines, and give each ZooKeeper server around 1GB of RAM, and if possible, its own dedicated disk. For very heavily loaded clusters, run ZooKeeper servers on separate machines from the Region Servers (DataNodes and TaskTrackers).

To point HBase at an existing ZooKeeper cluster, add a suitably configured zoo.cfg to the CLASSPATH. HBase will see this file and use it to figure out where ZooKeeper is. Additionally set HBASE_MANAGES_ZK in ${HBASE_HOME}/conf/hbase-env.sh to false so that HBase doesn't mess with your ZooKeeper setup:

```
   ...
  # Tell HBase whether it should manage it's own instance of Zookeeper or not.
  export HBASE_MANAGES_ZK=false
```

As an example, to have HBase manage a ZooKeeper quorum on nodes *rs{1,2,3,4,5}.example.com*, bound to port 2222 (the default is 2181), use:

```
  ${HBASE_HOME}/conf/hbase-env.sh:


       ...
      # Tell HBase whether it should manage it's own instance of Zookeeper or not.
      export HBASE_MANAGES_ZK=true


  ${HBASE_HOME}/conf/hbase-site.xml:


  <configuration>
    ...
    <property>
      <name>hbase.zookeeper.property.clientPort</name>
      <value>2222</value>
      <description>Property from ZooKeeper's config zoo.cfg.
      The port at which the clients will connect.
      </description>
    </property>
    ...
    <property>
      <name>hbase.zookeeper.quorum</name>

<value>rs1.example.com,rs2.example.com,rs3.example.com,rs4.example.com,rs5.example.co
m</value>
      <description>Comma separated list of servers in the ZooKeeper Quorum.
      For example, "host1.mydomain.com,host2.mydomain.com,host3.mydomain.com".
      By default this is set to localhost for local and pseudo-distributed modes
      of operation. For a fully-distributed setup, this should be set to a full
      list of ZooKeeper quorum servers. If HBASE_MANAGES_ZK is set in hbase-env.sh
      this is the list of servers which we will start/stop ZooKeeper on.
      </description>
    </property>
    ...
  </configuration>
```

When HBase manages ZooKeeper, it will start/stop the ZooKeeper servers as a part of the regular start/stop scripts. If you would like to run it yourself, you can do:

```
${HBASE_HOME}/bin/hbase-daemons.sh {start,stop} zookeeper
```

If you do let HBase manage ZooKeeper for you, make sure you configure where it's data is stored. By default, it will be stored in /tmp which is sometimes cleaned in live systems. Do modify this configuration:

```
    <property>
      <name>hbase.zookeeper.property.dataDir</name>
      <value>${hbase.tmp.dir}/zookeeper</value>
      <description>Property from ZooKeeper's config zoo.cfg.
      The directory where the snapshot is stored.
      </description>
    </property>
```

Note that you can use HBase in this manner to spin up a ZooKeeper cluster, unrelated to HBase. Just make sure to set HBASE_MANAGES_ZK to false if you want it to stay up so that when HBase shuts down it doesn't take ZooKeeper with it.

For more information about setting up a ZooKeeper cluster on your own, see the ZooKeeper Getting Started Guide. HBase currently uses ZooKeeper version 3.2.2, so any cluster setup with a 3.x.x version of ZooKeeper should work.

Of note, if you have made *HDFS client configuration* on your Hadoop cluster, HBase will not see this configuration unless you do one of the following:

- Add a pointer to your HADOOP_CONF_DIR to CLASSPATH in hbase-env.sh.
- Add a copy of hdfs-site.xml (or hadoop-site.xml) to ${HBASE_HOME}/conf, or
- if only a small set of HDFS client configurations, add them to hbase-site.xml.

An example of such an HDFS client configuration is dfs.replication. If for example, you want to run with a replication factor of 5, hbase will create files with the default of 3 unless you do the above to make the configuration available to HBase.

Running and Confirming Your Installation

If you are running in standalone, non-distributed mode, HBase by default uses the local filesystem.

If you are running a distributed cluster you will need to start the Hadoop DFS daemons and ZooKeeper Quorum before starting HBase and stop the daemons after HBase has shut down.

Start and stop the Hadoop DFS daemons by running ${HADOOP_HOME}/bin/start-dfs.sh. You can ensure it started properly by testing the put and get of files into the Hadoop filesystem. HBase does not normally use the mapreduce daemons. These do not need to be started.

Start up your ZooKeeper cluster.

Start HBase with the following command:

```
${HBASE_HOME}/bin/start-hbase.sh
```

Once HBase has started, enter ${HBASE_HOME}/bin/hbase shell to obtain a shell against HBase from which you can execute commands. Type 'help' at the shells' prompt to get a list of commands. Test your running install by creating tables, inserting content, viewing content, and then dropping your tables. For example:

```
hbase> # Type "help" to see shell help screen
hbase> help
hbase> # To create a table named "mylittletable" with a column family of "mylittlecolumnfamily", type
hbase> create "mylittletable", "mylittlecolumnfamily"
hbase> # To see the schema for you just created "mylittletable" table and its single "mylittlecolumnfamily", type
hbase> describe "mylittletable"
hbase> # To add a row whose id is "myrow", to the column "mylittlecolumnfamily:x" with a value of 'v', do
hbase> put "mylittletable", "myrow", "mylittlecolumnfamily:x", "v"
hbase> # To get the cell just added, do
hbase> get "mylittletable", "myrow"
hbase> # To scan you new table, do
hbase> scan "mylittletable"
```

To stop HBase, exit the HBase shell and enter:

```
${HBASE_HOME}/bin/stop-hbase.sh
```

If you are running a distributed operation, be sure to wait until HBase has shut down completely before stopping the Hadoop daemons.

The default location for logs is ${HBASE_HOME}/logs.

HBase also puts up a UI listing vital attributes. By default its deployed on the master host at port 60010 (HBase RegionServers listen on port 60020 by default and put up an informational http server at 60030).

Upgrading

After installing a new HBase on top of data written by a previous HBase version, before starting your cluster, run the ${HBASE_DIR}/bin/hbase migrate migration script. It will make any adjustments to the filesystem data under hbase.rootdir necessary to run the HBase version. It does not change your install unless you explicitly ask it to.

Example API Usage

For sample Java code, see org.apache.hadoop.hbase.client documentation.

If your client is NOT Java, consider the Thrift or REST libraries.

2. Maven 2

Maven is a high-level, intelligent project management, build and deployment tool provided by Apache's software foundation group. Maven deals with application development lifecycle management. Maven was originally developed to manage and to minimize the complexities of building the Jakarta Turbine project. But its powerful capabilities have made it a core entity of the Apache Software Foundation projects. Actually, for a long time there was a need to standardized project development lifecycle management system and Maven has emerged as a perfect option that meets the needs. Maven has become the de-facto build system in many open source initiatives and it is rapidly being adopted by many software development organizations. This tutorial provides you introduction to Maven 2, shifting from Ant to Maven, Maven basics, Installing and working with maven 2, Maven plugins etc.

Introduction to Maven 2

Maven2 is an Open Source build tool that made the revolution in the area of building projects. Like the build systems as "**make**" and "**ant**" it is not a language to combine the build components but it is a build lifecycle framework. A development team does not require much time to automate the project's build infrastructure since maven uses a standard directory layout and a default build lifecycle. Different development teams, under a common roof can set-up the way to work as standards in a very short time. This results in the automated build infrastructure in more stable state. On the other hand, since most of the setups are simple and reusable immediately in all the projects using maven therefore many important reports, checks, build and test animation are added to all the projects. Which was not possible without maven because of the heavy cost of every project setup?

Maven 2.0 was first released on 19 October 2005 and it is not backward compatible with the plugins and the projects of maven1. In December 2005, a lot of plugins were added to maven but not all plugins that exists for maven1 are ported yet. Maven 2 is expected to stabilize quickly with most of the Open Source technologies. People are introduced to use maven as the core build system for Java development in one project and a multi-project environment. After a little knowledge about the maven, developers are able to setup a new project with maven and also become aware of the default maven project structure. Developers are easily enabled to configure maven and its plugins for a project. Developers enable common settings for maven and its plugins over multiple projects, how to generate, distribute and deploy products and reports with maven so that they can use repositories to set up a company repository. Developers can also know about the most important plugins about how to install, configure and use them, just to look for other plugins to evaluate them so that they can be integrated in their work environment.

Maven is the standard way to build projects and it also provides various other characters like clearing the definition of the project, ways to share jars across projects. It also provides the easy way to publish project information (OOS).

Originally maven was designed to simplify the building processes in the Jakarta Turbine project. Several projects were there containing their own slightly different Ant build files and JARs were checked into CVS. An apache group's tool that can build the projects, publish project information, defines what the project consists of and that can share JARs across several projects. The result of all these requirement was the maven tool that builds and manages the java-based-project.

Maven 2: Features

Maven is a high-level, intelligent project management, build and deployment tool provided by Apache's software foundation group. Maven deals with application development lifecycle management. Maven was originally developed to manage and to minimize the complexities of building the Jakarta Turbine project. But its powerful capabilities have made it a core entity of the Apache Software Foundation projects. Actually, for a long time there was a need to standardized project development lifecycle management system and Maven has emerged as a perfect option that meets the needs. Maven has become the de-facto build system in many open source initiatives and it is rapidly being adopted by many software development organizations.

Maven was borne of the very practical desire to make several projects at Apache work in a consistence manner. So that developers could freely move between these projects, knowing clearly how they all worked by understanding how one of them worked.

If a developer spent time understanding how one project built it was intended that they would not have to go through this process again when they moved on to the next project. The same idea extends to testing, generating documentation, generating metrics and reports, testing and deploying. All projects share enough of the same characteristics, an understanding of which Maven tries to harness in its general approach to project management.

On a very high level all projects need to be built, tested, packaged, documented and deployed. There occurs infinite variation in each of the above mentioned steps, but these variation still occur within the confines of a well defined path and it is this path that Maven attempts to present to everyone in a clear way. The easiest way to make a path clear is to provide people with a set of patterns that can be shared by anyone involved in a project.

The key benefit of this approach is that developers can follow one consistent build lifecycle management process without having to reinvent such processes again. Ultimately this makes developers more productive, agile, disciplined, and focused on the work at hand rather than spending time and effort doing grunt work understanding, developing, and configuring yet another non-standard build system.

Maven: Features

1. **Portable:** Maven is portable in nature because it includes:
    - Building configuration using maven are portable to another machine, developer and architecture without any effort
    - **Non trivial:** Maven is non trivial because all file references need to be relative, environment must be completely controlled and independent from any specific file system.
2. **Technology:** Maven is a simple core concept that is activated through IoC container (Plexus). Everything is done in maven through plugins and every plugin works in isolation (ClassLoader). Plugings are downloaded from a plugin-repository on demand.

Maven's Objectives:

The primary goal of maven is to allow the developers to comprehend the complete state of a project in the shortest time by using easy build process, uniform building system, quality project management information (such as change Log, cross-reference, mailing lists, dependencies, unit test reports, test coverage reports and many more), guidelines for best practices and transparent migration to new features. To achieve to this goal Maven attempts to deal with several areas like:

- It makes the build process easy
- Provides a uniform building system
- Provides quality related project information
- Provides guidelines related to development to meet the best goal.
- Allows transparent migration to new features.

Shifting from Ant to Maven

Maven is entirely a different creature from Ant. Ant is simply a toolbox whereas Maven is about the application of patterns in order to achieve an infrastructure which displays the characteristics of visibility, reusability, maintainability, and comprehensibility. It is wrong to consider Maven as a build tool and just a replacement for Ant.

Ant Vs Maven

There is nothing that Maven does that Ant cannot do. Ant gives the ultimate power and flexibility in build and deployment to the developer. But Maven adds a layer of abstraction above Ant (and uses Jelly). Maven can be used to build any Java application. Today JEE build and deployment has become much standardized. Every enterprise has some variations, but in general it is all the same: deploying EARs, WARs, and EJB-JARs. Maven captures this intelligence and lets you achieve the build and deployment in about 5-6 lines of Maven script compared to dozens of lines in an Ant build script.

Ant lets you do any variations you want, but requires a lot of scripting. Maven on the other hand mandates certain directories and file names, but it provides plugins to make life easier. The restriction imposed by Maven is that only one artifact is generated per project (A project in Maven terminology is a folder with a project.xml file in it). A Maven project can have sub projects. Each sub project can build its own artifact. The topmost project can aggregate the artifacts into a larger one. This is synonymous to jars and wars put together to form an EAR. Maven also provides inheritance in projects.

Maven : Stealing the show

Maven simplifies build enormously by imposing certain fixed file names and acceptable restrictions like one artifact per project. Artifacts are treated as files on your computer by the build script. Maven hides the fact that everything is a file and forces you to think and script to create a deployable artifact such as an EAR. Artifact has a dependency on a particular version of a third party library residing in a shared remote (or local) enterprise repository, and then publish your library into the repository as well for others to use. Hence there are no more classpath issues. No more mismatch in libraries. It also gives the power to embed even the Ant scripts within Maven scripts if absolutely essential.

Common concerns to build a project

Some of the common concerns while building a project are:

1. **Project directory structure:** The directory structure of a Web application project is different from that of an EJB application project. Similarly the output of a Web application project is typically a WAR file while that of an EJB application is a JAR file.

2. **Directory naming conventions:** For a specific project type, the typical requirements in terms of directory layout and naming conventions are almost the same. Without a unified framework such as Maven, developers mostly spend time in configuring such nitty details like setting up directories for source, resources, test case source, testing time resources, classes, and project dependencies.

3. **The build output:** Developers spend a good chunk of time in creating build scripts such as ANT scripts to execute build tasks according to the project layout. This entire endeavor ends up being chaotic and in a large-scale project it can lead to a maintenance nightmare demanding dedicated resources just to focus on such build aspects.

5. Maven basics

A Project Object Model or POM is the fundamental unit of work in Maven. It is an xml file that contains information about the project and configuration details used by Maven to build the project. It contains default values for most projects.

I . Project Object Model

A Project Object Model or POM is the fundamental unit of work in Maven. It is an xml file that contains information about the project and configuration details used by Maven to build the project. It contains default values for most projects. For examples
the build directory, which is "target";
the source directory, which is "src/main/java";
the test source directory, which is "src/main/test"; and so on.

POM also contains the goals and plugins. So, while executing a task or goal, Maven looks for the POM in the current directory. It reads the POM, gets the needed configuration information, then executes the goal.

Some of the configuration that can be specified in the POM are the project dependencies, the plugins or goals that can be executed, the build profiles, and so on. Other information such as the project version, description, developers, mailing lists and such can also be specified.

The minimum requirement for a POM are the following:

- project root
- modelVersion - should be set to 4.0.0
- groupId - the id of the project's group.
- artifactId - the id of the artifact (project)
- version - the version of the artifact under the specified group

Here's an example:

```
<project>
  <modelVersion>4.0.0</modelVersion>
  <groupId>net.roseindia.maven</groupId>
  <artifactId>HelloMaven</artifactId>
  <version>1</version>
</project>
```

Learn to Set Up An Internal Private Repository For An Organization

We have set up an internal Maven Repository for our organisation so that the developers are not wasting time in searching and downloading the required project libraries. This also allows us to have a single company wide repository for project artifacts. The setup steps are not too much complicated but we didn't run into several issues while setup the local repository server (artifactory), for the first time.

When you're using maven at your company, you almost always want to setup local maven repository as relying on ibiblio for nightly / production builds is not a great idea and it takes time to download the library files if your development team is big. When setting up a local repository you don't want to setup the entire ibiblio repository locally as it is huge and has more libraries than you'll ever be using for your project. Maven-repository (in our case maven artifactory) is a repository server, which acts as your internal maven repository and downloads jars from ibiblio or other public maven repositories on demand and store it for further use in the project builds. And all this is transparent to the developer running a maven build.

The other neat thing about local maven-repository is that it allows you to neatly separate and organize jars that might not be available on ibiblio i.e. the 3rd-party artifacts (some company specific shared library or a commercial library).

Software requirements to set up the maven repository:

- **Artifactory:** Download and install the artifactory from the site http://www.jfrog.org/sites/artifactory/latest/. Artifactory comes with the application that can be installed into Tomcat.

- **JDK 1.6:** Get the information for downloading and installation from the site http://www.jfrog.org/sites /artifactory/latest/install .html.
- **Tomcat 6.0**

A Quick glance to the steps how we set up our local maven repository :

1. Download the appropriate Artifactory.zip file. You can get it from http://www.jfrog.org/sites/artifactory/latest/. Download and unzip the file in your directory of choice. We have downloaded artifactory-1.2.1.zip at our end.
2. Lets take the Installation directory as **D:\artifactory-1.2.2\artifactory-1.2.2**, Extract the artifactory-1.2.1.zip into the <artifactory-install-directory> directory.
3. Create repository configuration file artifactory.config.xml into <artifactory-install-directory>/etc/ directory and paste the following content in the artifactory.config.xml file:

```
<config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://artifactory.jfrog.org/xsd/1.0.0"
xsi:schemaLocation="http://artifactory.jfrog.org/xsd/1.0.0
http://www.jfrog.org/xsd/artifactory-v1_0_0.xsd">
<!-- Backup every 12 hours -->
<!--<backupCronExp>0 0 /12 * * ?</backupCronExp>-->
<localRepositories>

<localRepository>
<key>private-internal-repository</key>
<description>Private internal repository</description>
<handleReleases>true</handleReleases>
<handleSnapshots>true</handleSnapshots>
</localRepository>

<localRepository>
<key>3rd-party</key>
<description>3rd party jars added manually</description>
<handleReleases>true</handleReleases>
<handleSnapshots>false</handleSnapshots>
</localRepository>
```

```
</localRepositories>

<remoteRepositories>

<remoteRepository>
<key>ibiblio</key>
<handleReleases>true</handleReleases>
<handleSnapshots>false</handleSnapshots>
<excludesPattern>org/artifactory/**,org/jfrog/**</excludesPattern>
<url>http://repo1.maven.org/maven2</url>
<proxyRef>proxy1</proxyRef>
</remoteRepository>

<remoteRepository>
<key>ibiblio.org</key>
<description>ibiblio.org</description>
<handleReleases>true</handleReleases>
<handleSnapshots>false</handleSnapshots>
<excludesPattern>org/artifactory/**</excludesPattern>
<url>http://www.ibiblio.org/maven2</url>
<proxyRef>proxy1</proxyRef>
</remoteRepository>

<remoteRepository>
<key>java.net</key>
<description>java.net</description>
<handleReleases>true</handleReleases>
<handleSnapshots>false</handleSnapshots>
<excludesPattern>org/artifactory/**,org/jfrog/**</excludesPattern>
<url>http://download.java.net/maven/2</url>
<proxyRef>proxy1</proxyRef>
</remoteRepository>

</remoteRepositories>

<proxies>
```

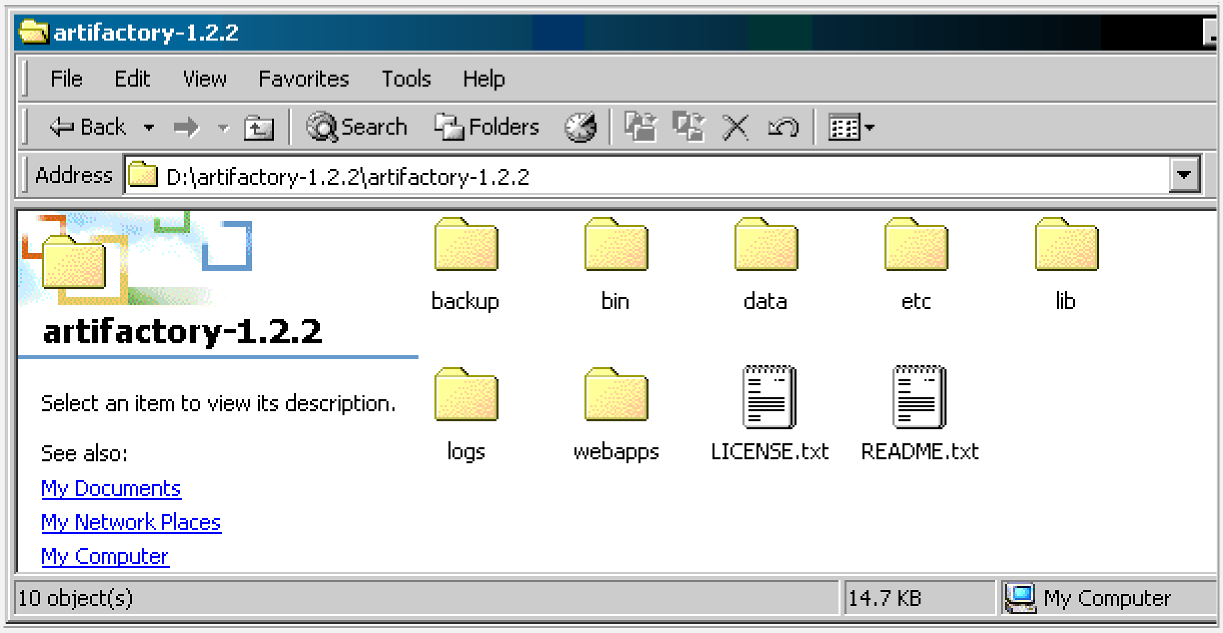
```
<proxy>
<key>proxy1</key>
<host>192.168.10.80</host>
<port>9090</port>
<username></username>
<password></password>
<domain>192.168.10.80</domain>
</proxy>
</proxies>
</config>
```

4. At our end we are using squid proxy server to connect with the internet. Our artifactory repository server will connect to internet through squid proxy server running on the machine 192.168.10.80 at port 9090. You can also use artifactory server without proxy. We have made two local and three remote repositories.

5. Download and install tomcat 6.0 on your machine.
6. Copy artifactor.war from artifactory-1.2.1\webapps to the webapps folder of your installed tomcat directory.
7. Specify the local artifactory installation folder to the tomcat environment. Go to Start-->Programs --> Apache Tomcat 6.0--> Configure Tomcat and specify the installation folder to the Java Options. -Dartifactory.home=<artifactory-install-directory>
8. Now start the tomcat and configure your clients to use maven artifactory repository.
9. To access the admin control panel type http://<server>:<port>/artifactory and login as User: admin and Password: password.

Directory Structure of Artifactory-1.2.2

Here are the folders that are shipped with the **Artifactory-1.2.2.zip** file.



- **bin:** batch files are used to run the included jetty web server.
- **lib:** Contains all the jar files required to run any application.
- **webapps:** Contains the war files of an application. We can also copy and install it in tomcat.
- **logs:** Includes all the log files.
- **backup:** Backs up the repository. We can use 'cron' expressions to setup the backup policy and Quartz scheduler to run the backup at the specified time. The backup interval is specified in the config.xml file inside the ‘ARTIFACTORY_INSTALLATION _FOLDER>/etc/artifactory folder’.
- **data:** Includes the derby database files. If you are interested to clean up the repository then all the things containing in this folder are deleted. In case of new installation process this folder is empty.
- **etc:** Includes the artifactory configuration files **"artifactory.config.xml"**, **"jetty.xml"** and **"log4j.properties"**.

Deployment in Tomcat 6.0

Deploy the **'war'** file of your application in ‘<ARTIFACTORY_INSTALLATION _FOLDER>/webapp’ to ‘<TOMCAT_INSTALLATION_FOLDER> /webapps’. There is no need to change the configuration with jdk1.6 and Tomcat 6.0. Tomcat detects the web application and deploy it.

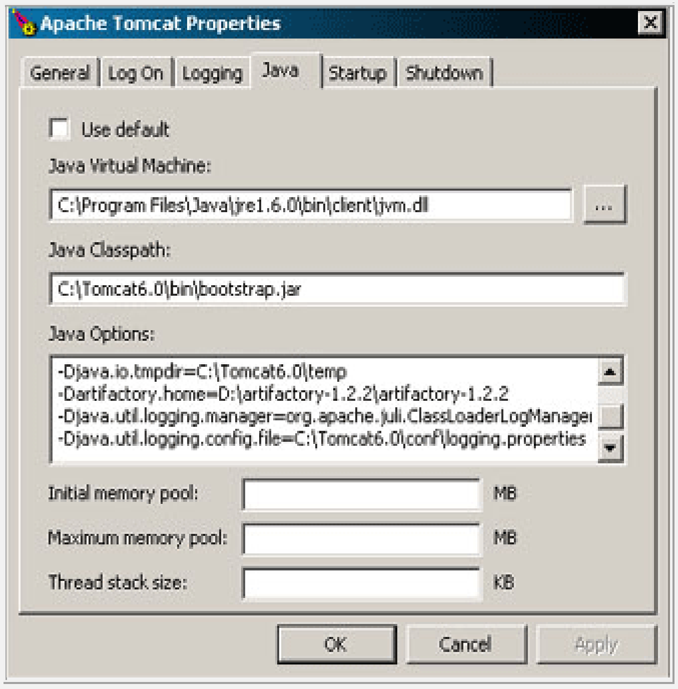
Once the application is deployed successfully, the web application requires the following information:

- Database location to store artifacts
- Artifactory config xml file location
- Backup folder location

To specify all the above three information, a single configuration is used. We only need to specify the artifactory installation folder location during Tomcat startup and artifactory does all the rest task by itself. There is another approach, that is, setup the connection to the derby database using jdbc and configure artifactory in the web application (by including artifactory.config.xml in the web application).

We can use the environment variable to specify the location of the artifactory installation folder. In case of Windows, we can add it to Tomcat startup options and Configure Tomcat by specifying the installation folder to the Java Options.

-Dartifactory.home=<artifactory-install-directory> e.g. **-Dartifactory.home=D:\artifactory-1.2.2\artifactory-1.2.2**



Use default

Java Virtual Machine: C:\Program Files\Java\jre1.6.0\bin\client\jvm.dll

Java Classpath: C:\Tomcat6.0\bin\bootstrap.jar

Java Options:

```
-Djava.io.tmpdir=C:\Tomcat6.0\temp
-Dartifactory.home=D:\artifactory-1.2.2\artifactory-1.2.2
-Djava.util.logging.manager=org.apache.juli.ClassLoaderLogManager
-Djava.util.logging.config.file=C:\Tomcat6.0\conf\logging.properties
```

Initial memory pool: MB

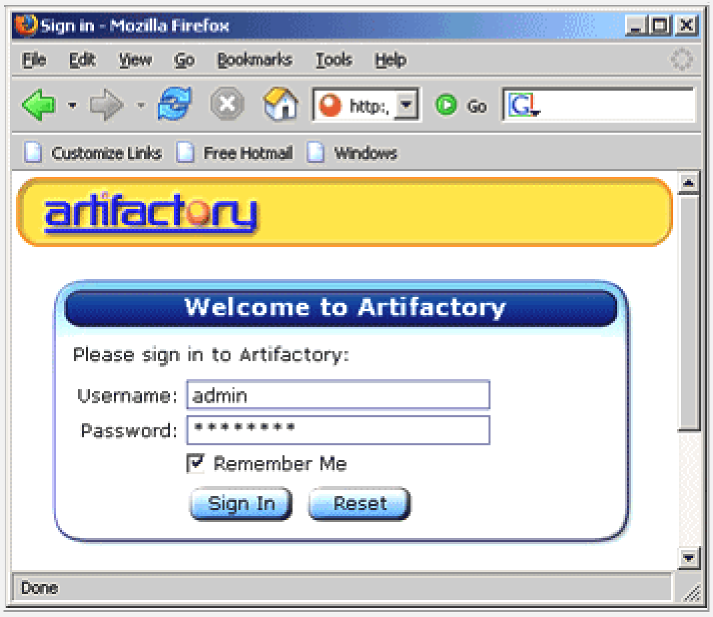
Maximum memory pool: MB

Thread stack size: KB

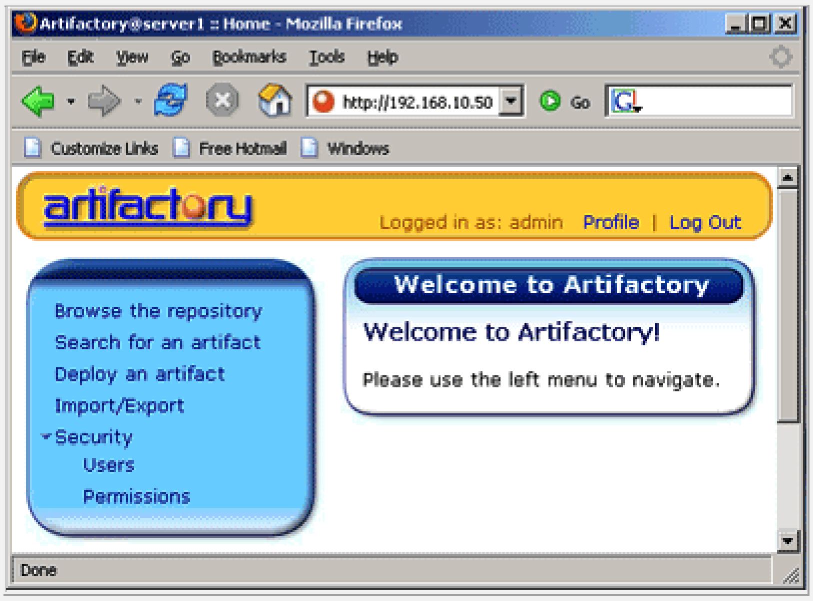
Navigating Artifactory

Start Tomcat, open the browser and navigate http://localhost:8080/artifactory

Here is the artifactory home page shown below:


Sign in - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
Customize Links Free Hotmail Windows
artifactory
Welcome to Artifactory
Please sign in to Artifactory:
Username: admin
Password: ********
Remember Me
Sign In
Reset
Done

Sign in username as **'admin'** and password as **'password'.** You can view the content of the repository simply by clicking on the Repository Browser link.


Artifactory@server1 :: Home - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
http://192.168.10.50
Customize Links Free Hotmail Windows
artifactory
Logged in as: admin Profile | Log Out
Browse the repository
Search for an artifact
Deploy an artifact
Import/Export
Security
Users
Permissions
Welcome to Artifactory
Welcome to Artifactory!
Please use the left menu to navigate.
Done

V. Configuring maven to use the new repository:

You can use either of the settings.xml or the pom.xml files to configure maven to use the local repository.

- **Configure maven using settings.xml file:** Maven uses the settings.xml file contained in .m2 folder inside the C\Document and Setting\Administrator to get the location of the maven repository. In case of no repository is specified then maven uses the default repository from **ibiblio.org**. We will have to make changes in the settings.xml file to use the new repository. Here is the settings.xml shown below:

```
<profiles>
<profile>
```

```
<id>dev</id>
<properties>
<tomcat6x.home>C:/InstalledPrograms/apache-
tomcat-6.0</tomcat6x.home>
</properties>
<repositories>
<repository>
<id>central</id>
<url>http://localhost:8080/artifactory/repo</url>
<snapshots>
<enabled>false</enabled>
</snapshots>
</repository>
<repository>
<id>snapshots</id>
<url>http://localhost:8080/artifactory/repo</url>
<releases>
<enabled>false</enabled>
</releases>
</repository>
</repositories>
<pluginRepositories>
<pluginRepository>
<id>central</id>
<url>http://localhost:8080/artifactory/repo</url>
<snapshots>
<enabled>false</enabled>
</snapshots>
</pluginRepository>
<pluginRepository>
<id>snapshots</id>
<url>http://localhost:8080/artifactory/repo</url>
<releases>
<enabled>false</enabled>
</releases>
</pluginRepository>
</pluginRepositories>
</profile>
</profiles>
```

- **Configure maven using project "pom.xml" :** We can also set the repository settings through the pom.xml file. Simple pom.xml is shown below:

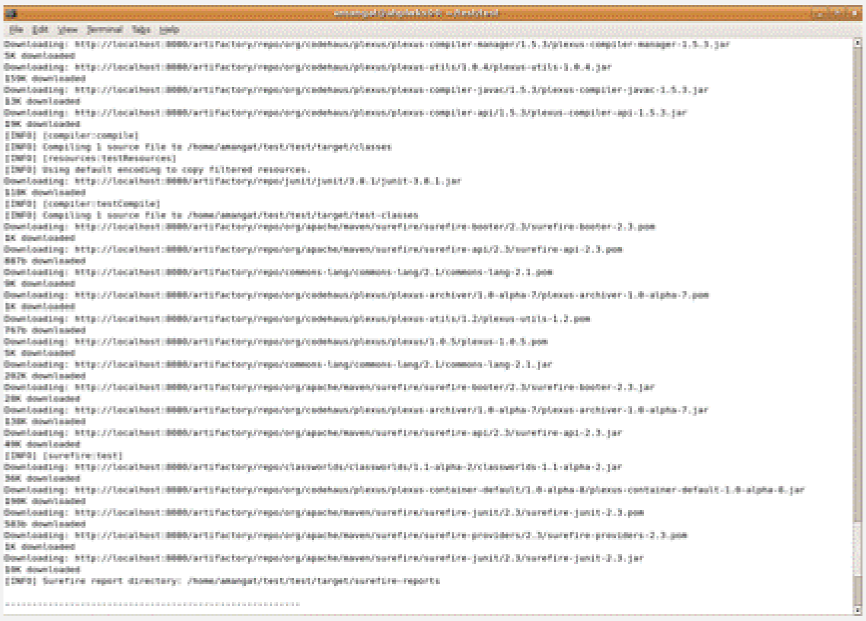
```
<project      xmlns="http://maven.apache.org/POM/4.0.0"
```

```
xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/maven-v4_0_0.xsd">
<modelVersion>4.0.0</modelVersion>
<groupId>test</groupId>
<artifactId>test</artifactId>
<packaging>jar</packaging>
<version>1.0-SNAPSHOT</version>
<name>test</name>
<url>http://maven.apache.org</url>
<repositories>
<repository>
<id>central</id>
<url>http://localhost:8080/artifactory/repo</url>
<snapshots>
<enabled>false</enabled>
</snapshots>
</repository>
<repository>
<id>snapshots</id>
<url>http://localhost:8080/artifactory/repo</url>
<releases>
<enabled>false</enabled>
</releases>
</repository>
</repositories>
<pluginRepositories>
<pluginRepository>
<id>central</id>
<url>http://localhost:8080/artifactory/repo</url>
<snapshots>
<enabled>false</enabled>
</snapshots>
</pluginRepository>
<pluginRepository>
<id>snapshots</id>
<url>http://localhost:8080/artifactory/repo</url>
<releases>
<enabled>false</enabled>
</releases>
</pluginRepository>
</pluginRepositories>
<dependencies>
```

```
<dependency>
<groupId>junit</groupId>
<artifactId>junit</artifactId>
<version>3.8.1</version>
<scope>test</scope>
</dependency>
</dependencies>
</project>
```

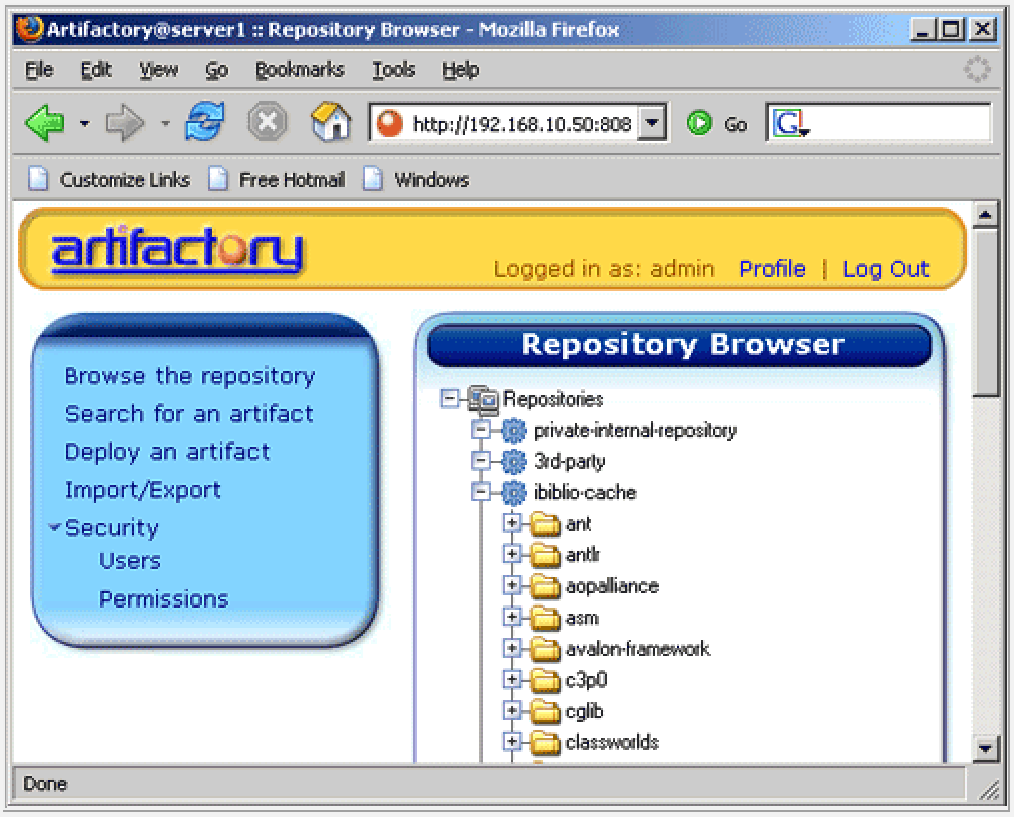
VI. Building using the new maven repository

While building the maven project, all the repositories should be downloaded using the new repository. The console will show the server maven uses as shown below:

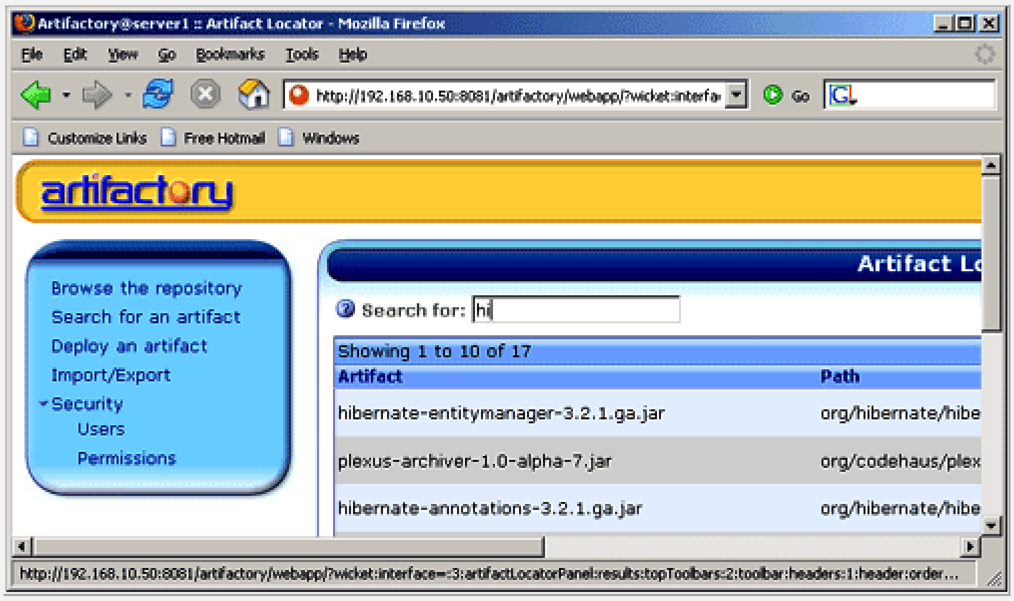


```
Downloading: http://localhost:8080/artifactory/repo/org/codehaus/plexus/plexus-compiler-manager/1.5.3/plexus-compiler-manager-1.5.3.jar
5K downloaded
Downloading: http://localhost:8080/artifactory/repo/org/codehaus/plexus/plexus-utils/1.0.4/plexus-utils-1.0.4.jar
159K downloaded
Downloading: http://localhost:8080/artifactory/repo/org/codehaus/plexus/plexus-compiler-javac/1.5.3/plexus-compiler-javac-1.5.3.jar
13K downloaded
Downloading: http://localhost:8080/artifactory/repo/org/codehaus/plexus/plexus-compiler-api/1.5.3/plexus-compiler-api-1.5.3.jar
19K downloaded
[INFO] [compiler:compile]
[INFO] Compiling 1 source file to /home/amangat/test/test/target/classes
[INFO] [resources:testResources]
[INFO] Using default encoding to copy filtered resources.
Downloading: http://localhost:8080/artifactory/repo/junit/junit/3.8.1/junit-3.8.1.jar
118K downloaded
[INFO] [compiler:testCompile]
[INFO] Compiling 1 source file to /home/amangat/test/test/target/test-classes
Downloading: http://localhost:8080/artifactory/repo/org/apache/maven/surefire/surefire-booter/2.3/surefire-booter-2.3.pom
1K downloaded
Downloading: http://localhost:8080/artifactory/repo/org/apache/maven/surefire/surefire-api/2.3/surefire-api-2.3.pom
887b downloaded
Downloading: http://localhost:8080/artifactory/repo/commons-lang/commons-lang/2.1/commons-lang-2.1.pom
9K downloaded
Downloading: http://localhost:8080/artifactory/repo/org/codehaus/plexus/plexus-archiver/1.0-alpha-7/plexus-archiver-1.0-alpha-7.pom
1K downloaded
Downloading: http://localhost:8080/artifactory/repo/org/codehaus/plexus/plexus-utils/1.2/plexus-utils-1.2.pom
767b downloaded
Downloading: http://localhost:8080/artifactory/repo/org/codehaus/plexus/plexus/1.0.5/plexus-1.0.5.pom
5K downloaded
Downloading: http://localhost:8080/artifactory/repo/commons-lang/commons-lang/2.1/commons-lang-2.1.jar
202K downloaded
Downloading: http://localhost:8080/artifactory/repo/org/apache/maven/surefire/surefire-booter/2.3/surefire-booter-2.3.jar
20K downloaded
Downloading: http://localhost:8080/artifactory/repo/org/codehaus/plexus/plexus-archiver/1.0-alpha-7/plexus-archiver-1.0-alpha-7.jar
138K downloaded
Downloading: http://localhost:8080/artifactory/repo/org/apache/maven/surefire/surefire-api/2.3/surefire-api-2.3.jar
40K downloaded
[INFO] [surefire:test]
Downloading: http://localhost:8080/artifactory/repo/classworlds/classworlds/1.1-alpha-2/classworlds-1.1-alpha-2.jar
36K downloaded
Downloading: http://localhost:8080/artifactory/repo/org/codehaus/plexus/plexus-container-default/1.0-alpha-8/plexus-container-default-1.0-alpha-8.jar
190K downloaded
Downloading: http://localhost:8080/artifactory/repo/org/apache/maven/surefire/surefire-junit/2.3/surefire-junit-2.3.pom
583b downloaded
Downloading: http://localhost:8080/artifactory/repo/org/apache/maven/surefire/surefire-providers/2.3/surefire-providers-2.3.pom
1K downloaded
Downloading: http://localhost:8080/artifactory/repo/org/apache/maven/surefire/surefire-junit/2.3/surefire-junit-2.3.jar
10K downloaded
[INFO] Surefire report directory: /home/amangat/test/test/target/surefire-reports

-------------------------------------------------------
```

Login into your new repository through the web browser and you see that the artifactory has downloaded and cached the artifacts from ibiblio.



It provides an advance search facility that can be used to search an artifact simply by using the advanced search technique.



Showing 1 to 10 of 17

Artifact	Path
hibernate-entitymanager-3.2.1.ga.jar	org/hibernate/hibe
plexus-archiver-1.0-alpha-7.jar	org/codehaus/plex
hibernate-annotations-3.2.1.ga.jar	org/hibernate/hibe

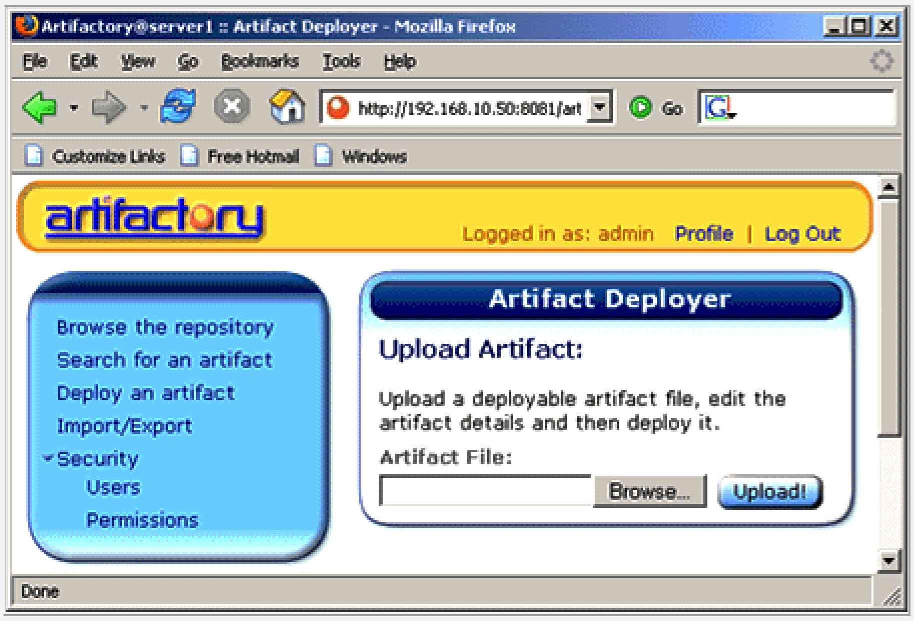
```
http://192.168.10.50:8081/artifactory/webapp/?wicket:interface=:3:artifactLocatorPanel:results:topToolbars:2:toolbar:headers:1:header:order...
```

VII. Installing artifacts to the repository :

We can use the **web UI** or **maven command line** to install the artifacts. Installation process through the web UI is simpler and faster and does not require any change in the configuration. While installation through the command line requires changes in the configuration in settings.xml and then we can use it in other scripts.

Installing artifacts using the web UI: Steps involved in the installation process are shown below:

1. Upload artifact (e.g. 'jar' file or 'pom' file) and deploy using Artifact Deployer.

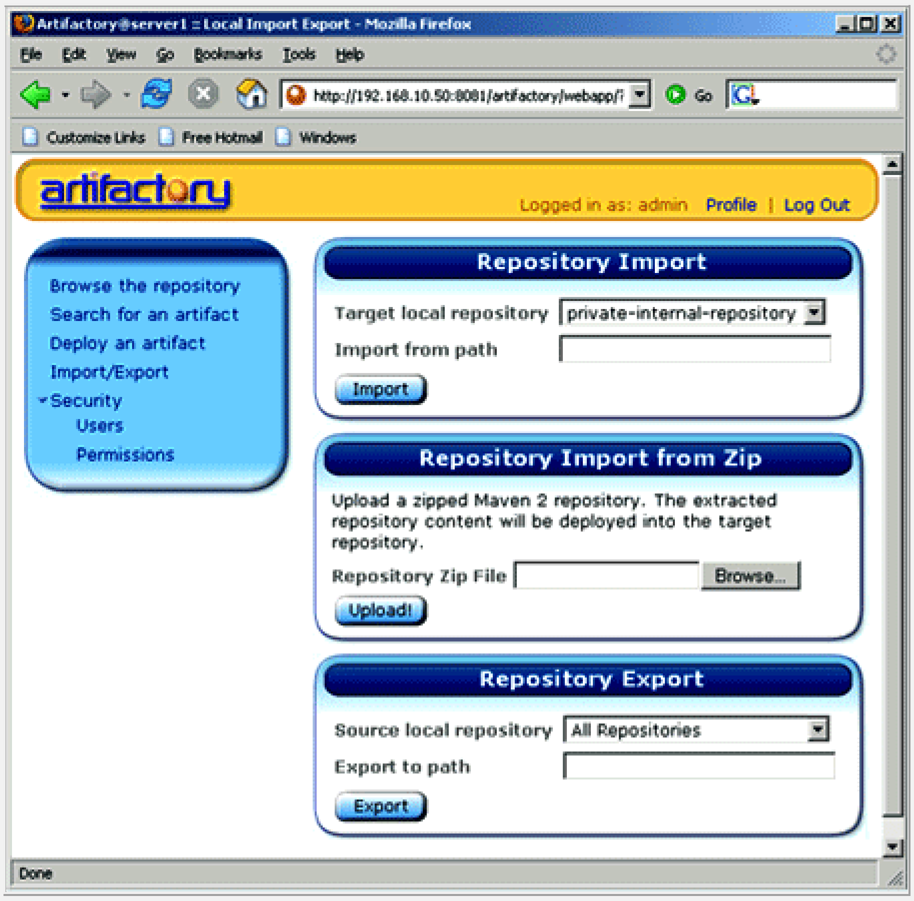


Upload Artifact:

Upload a deployable artifact file, edit the artifact details and then deploy it.

Artifact File:

2. If you upload the jar file then the artifactory will create the pom.xml file as well as you can also specify which repository to upload to.



Repository Import

Target local repository: private-internal-repository

Import from path

Repository Import from Zip

Upload a zipped Maven 2 repository. The extracted repository content will be deployed into the target repository.

Repository Zip File

Repository Export

Source local repository: All Repositories

Export to path

3. Once uploading is complete successfully, the artifact along with the ‘pom.xml’ file created by artifactory appears in the repository.

Installing artifacts from maven command line:
Using the command 'maven clean install' maven only packages and installs the artifact to the local repository. We need to add the additional configuration section in the **settings.xml** file to install it to the local internal repository. Steps involved in whole of the process are shown below:

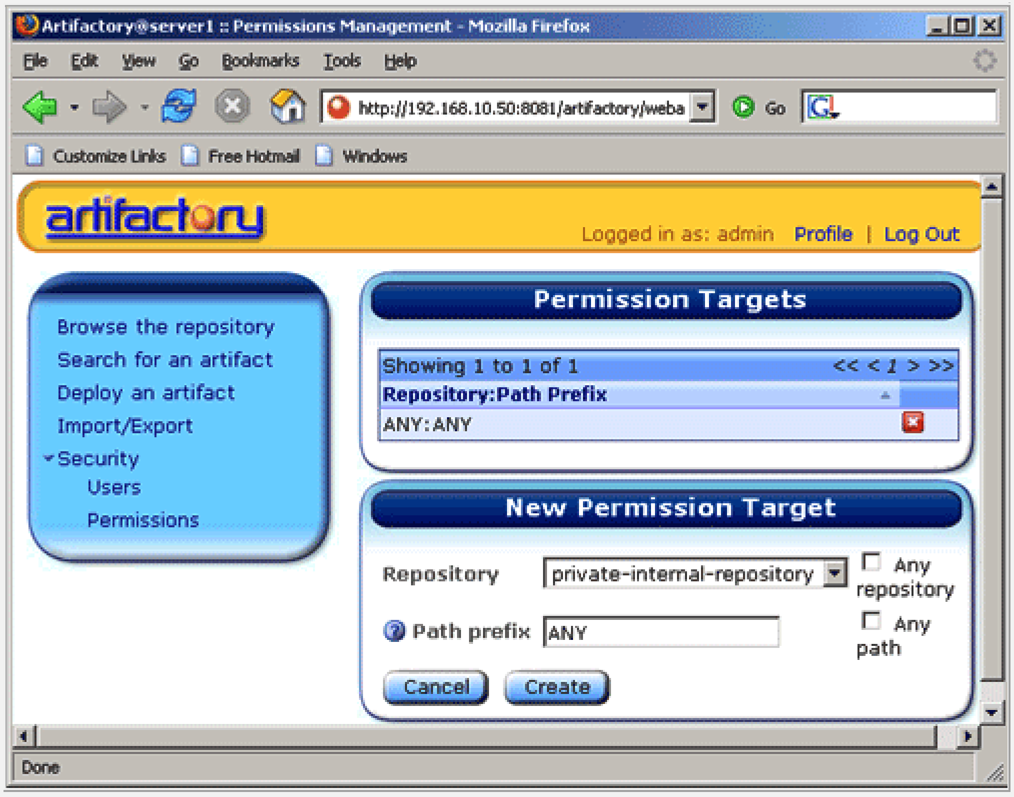
settings.xml

```
<settings>

<servers>
<server>
<id>organisation-internal</id>
<username>admin</username>
<password>password</password>
</server>
</servers>

</settings>
```

The command given below is used to install an artifact to internal maven repository.

```
mvn deploy:deploy-file -DrepositoryId=organisation-internal -
Durl=http://localhost:8080/artifactory/private-internal-repository
-DgroupId=test -DartifactId=test -Dversion=1.1 -Dpackaging=jar -
Dfile=target/test-1.1.jar
```

Both the **repository id** and the **server id** defined in the settings.xml should be same. The url should include the repository name to which the artifact is to be installed. The artifactory and the new artifacts appeared in the repository creates the 'pom' (pom.xml) file for us automatically.


Artifactory@server1 :: Permissions Management - Mozilla Firefox

Logged in as: admin Profile | Log Out

Permission Targets

Showing 1 to 1 of 1

Repository:Path Prefix
ANY:ANY

New Permission Target

Repository: private-internal-repository ☐ Any repository

Path prefix: ANY ☐ Any path

VIII. Other Artifactory features:

- **Backup the repository:** Backup policy is specified in the <ARTIFACTORY_INSTALLATION _FOLDER>/etc/artifactory .config.xml and 'cron' expression specifies the backup configuration. Backup configuration is shown below:

```
<config xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance"
xmlns="http://artifactory.jfrog.org/xsd/1.0.0"
xsi:schemaLocation="http://artifactory.jfrog.org/xsd/1.0.0
http://www.jfrog.org/xsd/artifactory-v1_0_0.xsd">
<!-- Backup every 12 hours -->
<backupCronExp>0 0 /12 * * ?</backupCronExp>
<localRepositories>
<localRepository>
<key>private-internal-repository</key>
<description>Private internal repository</description>
<handleReleases>true</handleReleases>
<handleSnapshots>true</handleSnapshots>
</localRepository>
<localRepository>
<key>3rd-party</key>
<description>3rd party jars added manually</description>
<handleReleases>true</handleReleases>
```

```
<handleSnapshots>false</handleSnapshots>
</localRepository>
</localRepositories>
<remoteRepositories>
<remoteRepository>
<key>ibiblio</key>
<handleReleases>true</handleReleases>
<handleSnapshots>false</handleSnapshots>
<excludesPattern>org/artifactory/**,org/jfrog/**</excludesPattern>
<url>http://repo1.maven.org/maven2</url>
</remoteRepository>
</remoteRepositories>
</config>
```

The directory '<ARTIFACTORY_INSTALLATION _FOLDER>/backups' contains the backups. The local repository on the developer's machine and the backups both have the same format. It allow us the repository contents to migrate easily to another implementation of maven repository.

Other features:

- Use the web UI to delete the artifacts
- Use the web UI to search for artifacts.
- Bulk import/export all artifacts in repository.
- If tomcat is not required then we can use the included jetty web server.

The overall conclusion is that an internal maven repository helps us to avoid conflicts due to different versions of libraries and it also speeds up the build process. Artifactory seems the better product among the 4 common maven repository. It has all the features that a good maven repository should have. The organization will not be locked into this tool since migration of the repository to another implementation is rather easy. A web UI simplifies the use of the repository even for the peoples who don't know the working of the repository.

8. Maven 2 Eclipse Plug-in

Plugins are great in simplifying the life of programmers; it actually reduces the repetitive tasks involved in the programming. In this article our experts will show you the steps required to download and install the Maven Plugin with your eclipse IDE.

Why Maven with Eclipse

Eclipse is an industry leader in IDE market, it is used very extensively in developing projects all around the world. Similarly, Maven is a high-level, intelligent project management, build and deployment tool provided by Apache's software foundation group. Maven deals with application development lifecycle management.

Maven–Eclipse Integration makes the development, testing, packaging and deployment process easy and fast. Maven Integration for Eclipse provides a tight integration for Maven into the IDE and avails the following features:

· It helps to launch Maven builds from within Eclipse

· It avails the dependency management for Eclipse build path based on Maven's pom.xml

· It resolves Maven dependencies from the Eclipse workspace without installing to local Maven repository

· It avails an automatic downloading of the required dependencies from the remote Maven repositories

· It provides wizards for creating new Maven projects, pom.xml or to enable Maven support on plain Java project

· It helps to search quickly for dependencies in Maven remote repositories

· It quickly fixes in the Java editor for looking up required dependencies/jars by the class or package name.

What do you Need?

1. Get the Eclipse Development Environment :
   In this tutorial we are using the eclipse-SDK-3.3-win32, which can be downloaded from http://www.eclipse.org/downloads/
2. Get Maven-eclipse-plugin-plugin :
   It is available at http://mevenide.codehaus.org/maven-eclipse-plugin-plugin/

Download and Install Eclipse

First download and install the eclipse plugin on your development machine then proceed with the installation process of the eclipse-maven plugin.

Steps to Install the eclipse-maven plugin

1.Open eclipse IDE and
go to **Help->Software Updates-> Find and Install**
as shown in Figure 1 below:

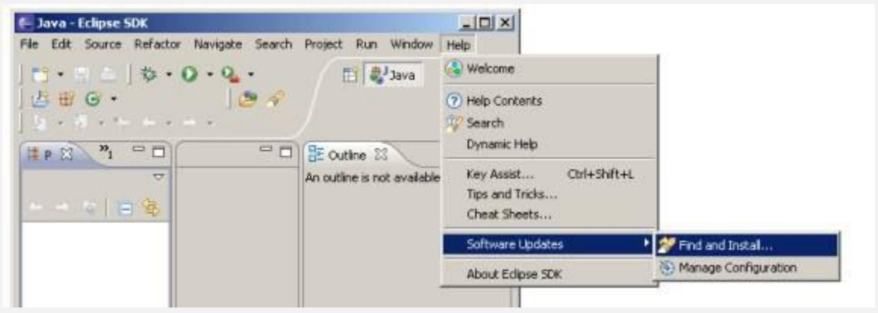
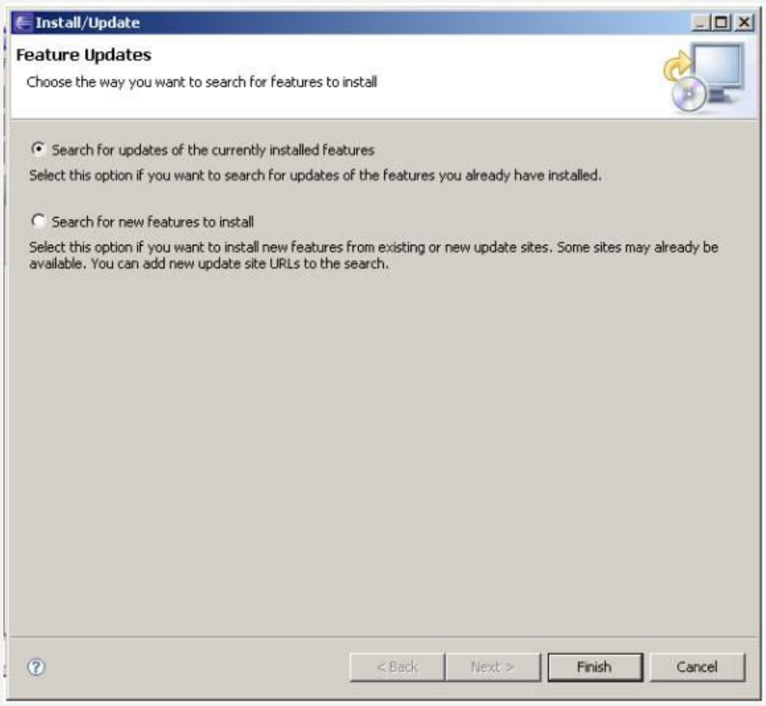


Figure 1

2. Choose the way you want to search for the features to Install in the **Install/update window**. It is show in the figure 2:



Install/Update

Feature Updates

Choose the way you want to search for features to install

[✓ Search for updates of the currently installed features]

Select this option if you want to search for updates of the features you already have installed.

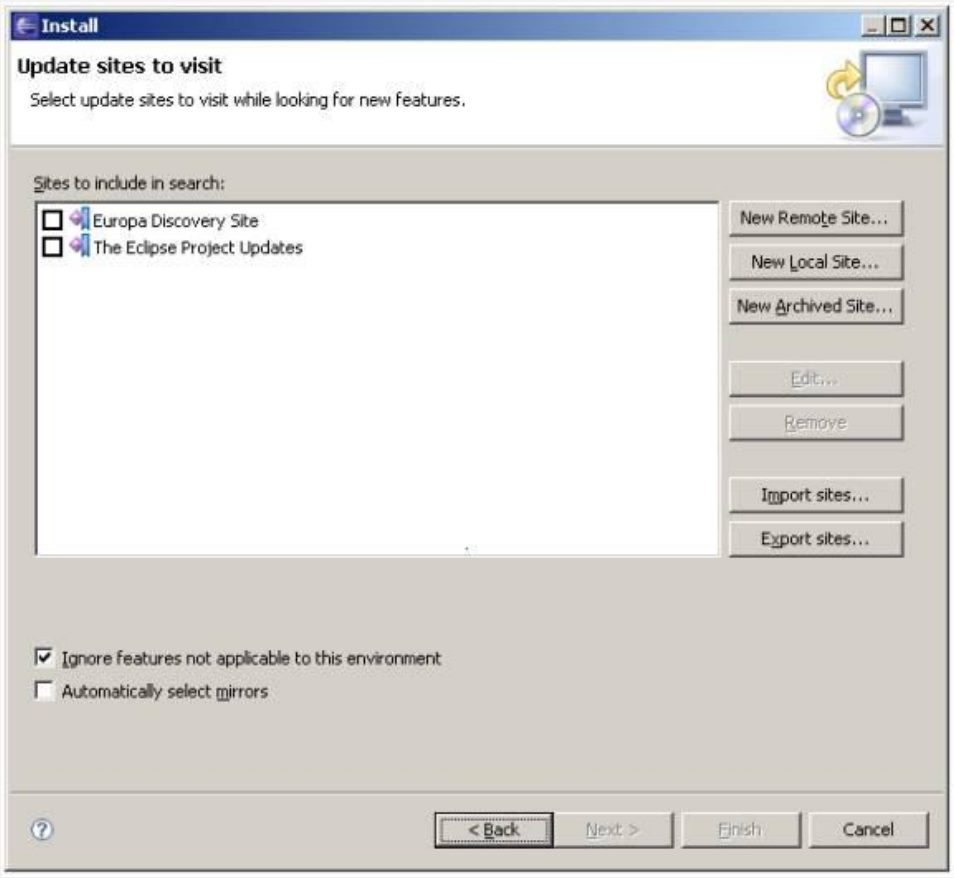
Search for new features to install

Select this option if you want to install new features from existing or new update sites. Some sites may already be available. You can add new update site URLs to the search.

Figure 2

Now, Select the option **“Search for new features to install**” and click on the “**Next**” button.

3. An Install wizard appears as shown in Figure 3 :



Install

Update sites to visit

Select update sites to visit while looking for new features.

Sites to include in search:

- [ ] Europa Discovery Site
- [ ] The Eclipse Project Updates

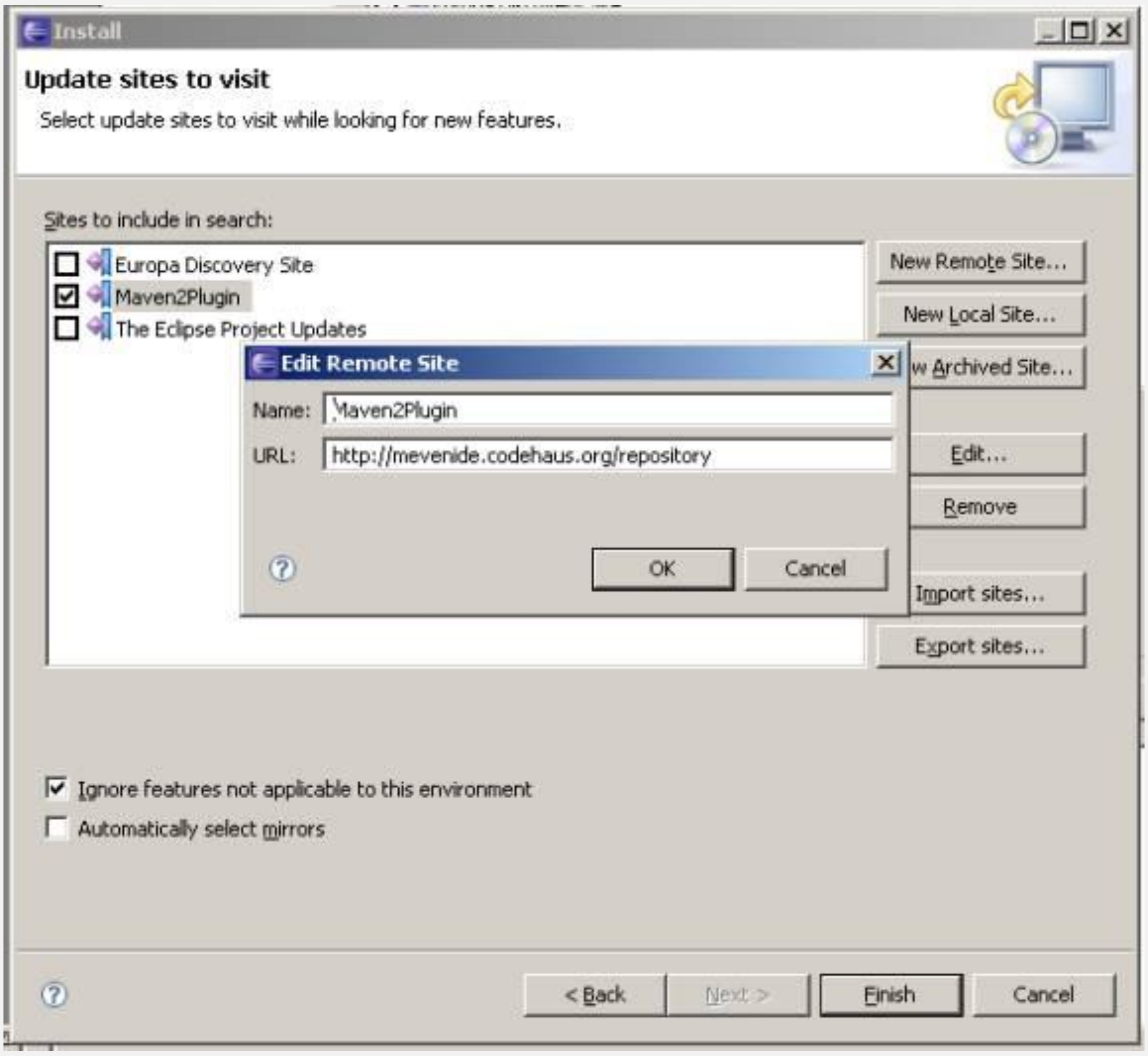
[✓ Ignore features not applicable to this environment]

Automatically select mirrors

Figure: 3

Now click on “**New Remote Site…**” and enter the following details in the dialog box:
Name**: Maven2Plugin**
URL**: http://mevenide.codehaus.org/repository**

As shown in figure 4



Name: Maven2Plugin
URL: http://mevenide.codehaus.org/repository

Figure 4
Now click on “**OK**” button.

4. Then update the information of the sites to visit for instance we have selected the **Maven2plugin** site in the Eclipse update list as shown in following figure (Figure 5)

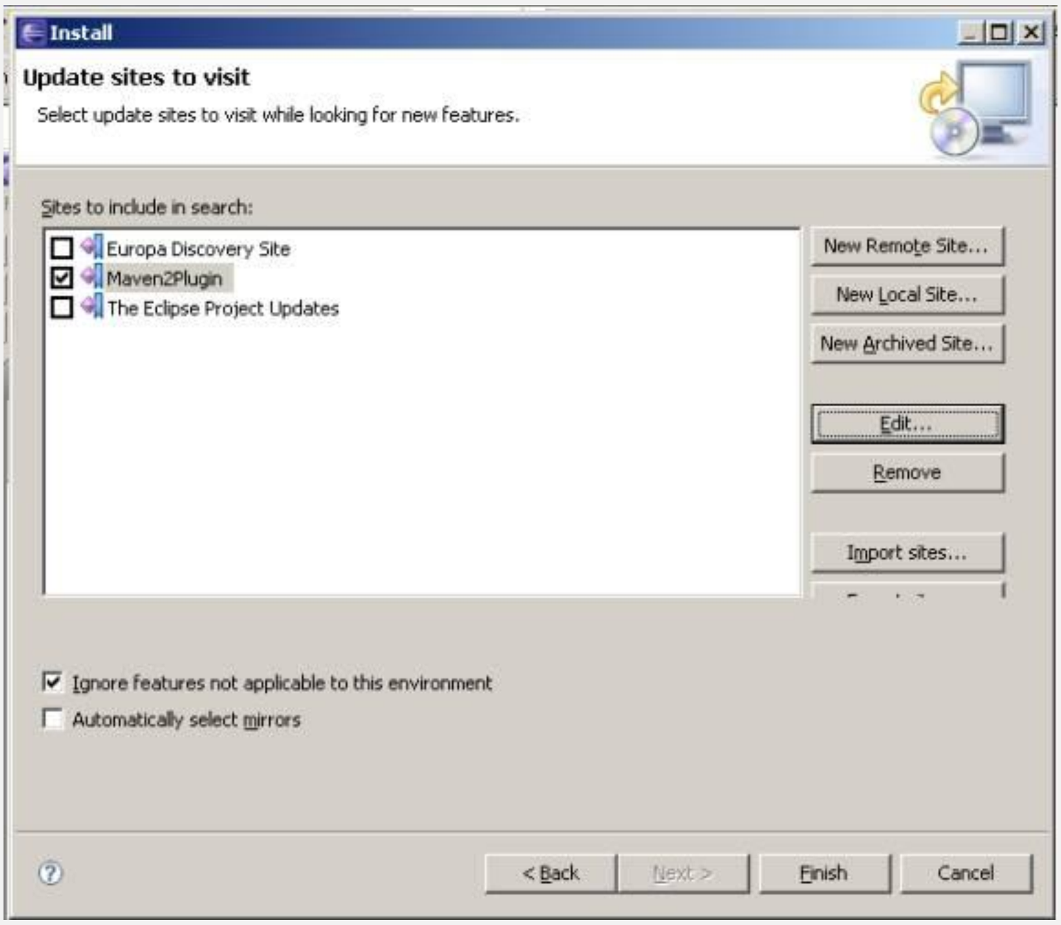

Install
Update sites to visit
Select update sites to visit while looking for new features.
Sites to include in search:
[ ] Europa Discovery Site
[✓] Maven2Plugin
[ ] The Eclipse Project Updates
New Remote Site...
New Local Site...
New Archived Site...
Edit...
Remove
Import sites...
[✓] Ignore features not applicable to this environment
[ ] Automatically select mirrors
< Back
Next >
Finish
Cancel

Figure 5

5. Click on the "**Finish**" button. Update manager starts searching for the updates as shown in Figure 6

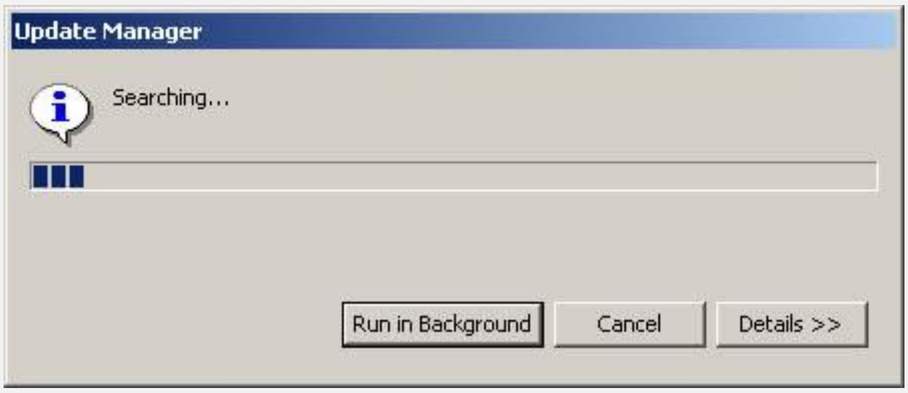
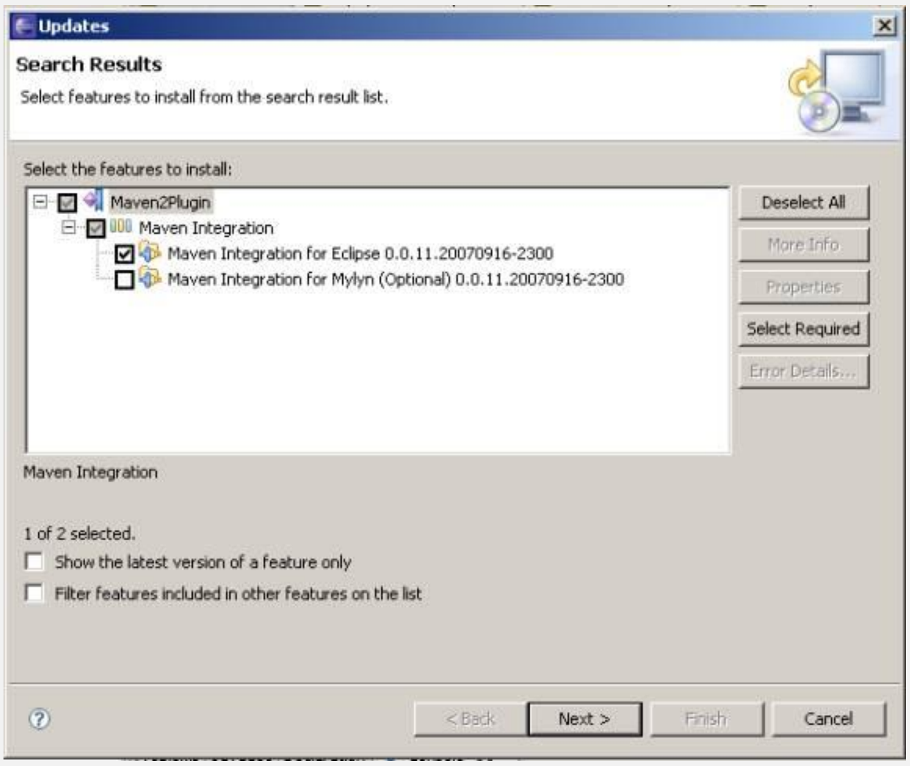


Figure 6

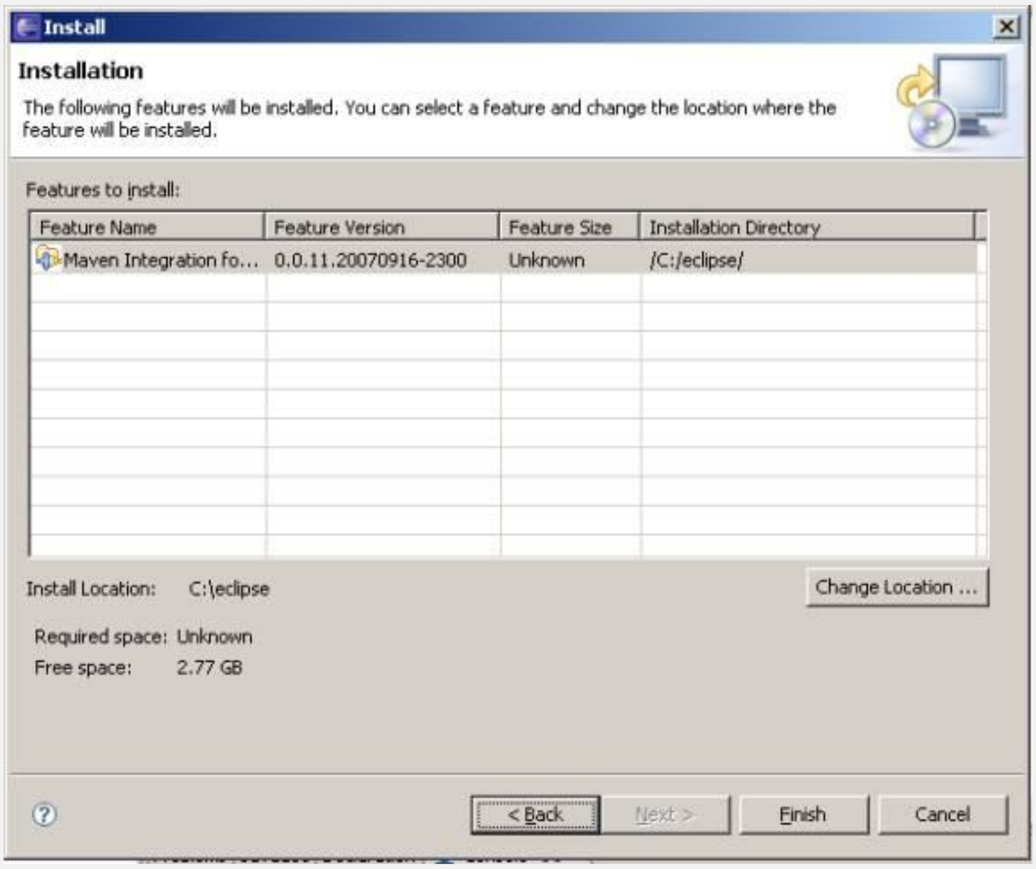
6. Then an Eclipse update window appears, select "**Maven2Plugin**" check box as show below:



Updates
Search Results
Select features to install from the search result list.
Select the features to install:
Maven2Plugin
Maven Integration
Maven Integration for Eclipse 0.0.11.20070916-2300
Maven Integration for Mylyn (Optional) 0.0.11.20070916-2300
Deselect All
More Info
Properties
Select Required
Error Details...
Maven Integration
1 of 2 selected.
Show the latest version of a feature only
Filter features included in other features on the list
< Back
Next >
Finish
Cancel

Click on the "**Next**" button and then Accept the terms and conditions in the next window and click the next button.

7. Then, An Installation confirmation window appears as shown below:



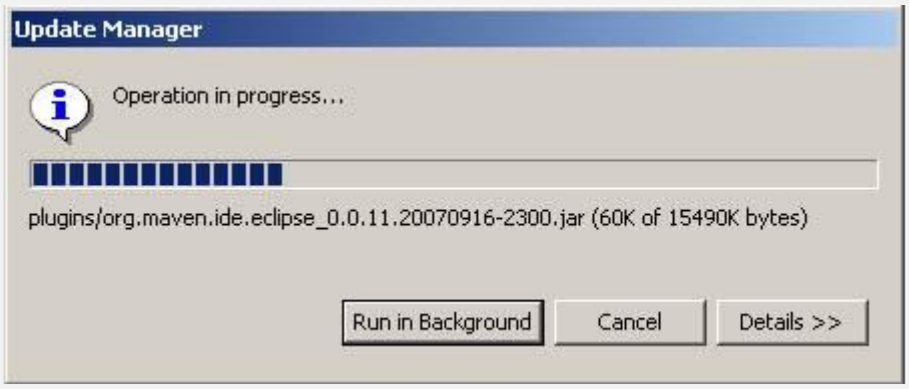
Install
Installation
The following features will be installed. You can select a feature and change the location where the feature will be installed.
Features to install:

Feature Name	Feature Version	Feature Size	Installation Directory
Maven Integration fo...	0.0.11.20070916-2300	Unknown	/C:/eclipse/

Install Location: C:\eclipse
Change Location ...
Required space: Unknown
Free space: 2.77 GB
< Back
Next >
Finish
Cancel

Click on "**Finish**" button to complete the installation process.

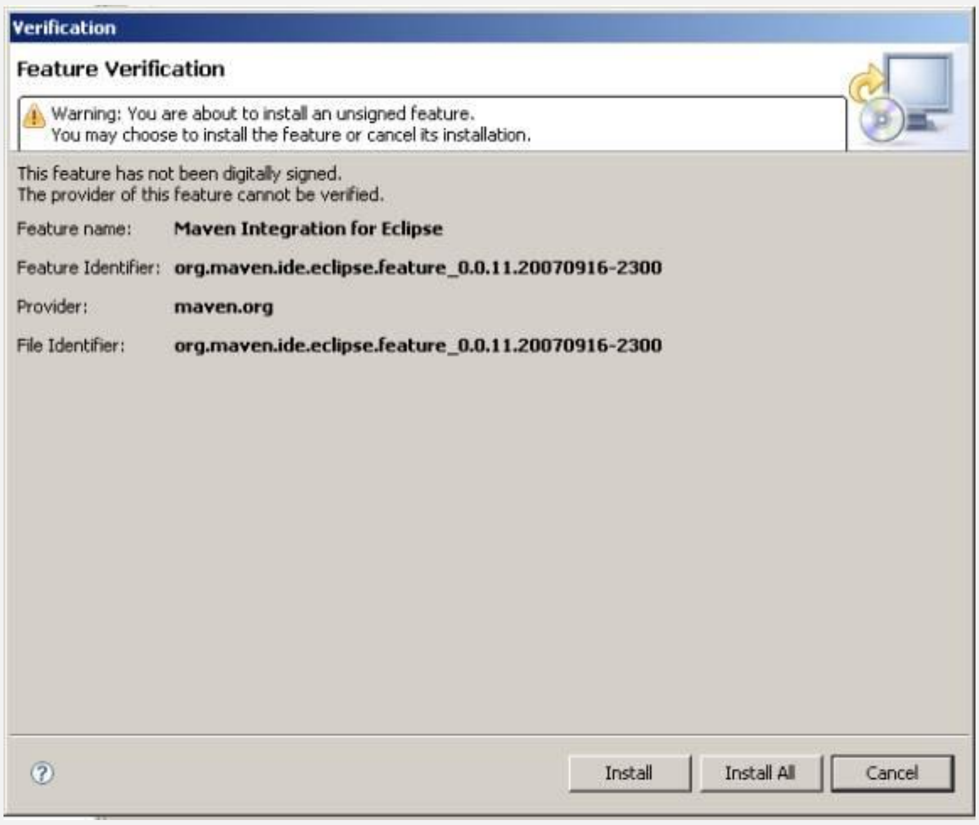
8. **Update manager** will download the files and install the Maven eclipse plugin for you.


Update Manager

Operation in progress...

plugins/org.maven.ide.eclipse_0.0.11.20070916-2300.jar (60K of 15490K bytes)

9. Then the installer displays the **Feature Verification** window as shown below:


Verification

Feature Verification

Warning: You are about to install an unsigned feature.
You may choose to install the feature or cancel its installation.

This feature has not been digitally signed.
The provider of this feature cannot be verified.

Feature name: Maven Integration for Eclipse

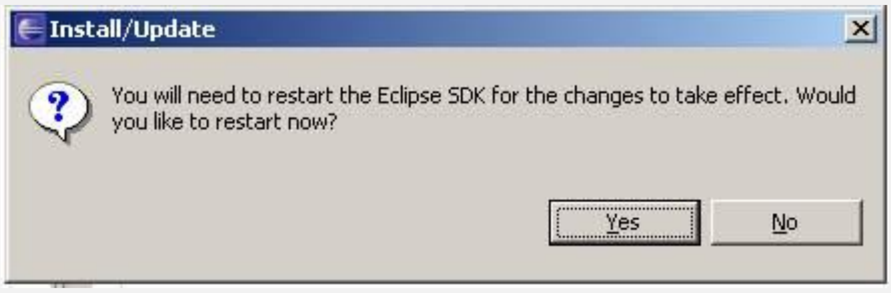
Feature Identifier: org.maven.ide.eclipse.feature_0.0.11.20070916-2300

Provider: maven.org

File Identifier: org.maven.ide.eclipse.feature_0.0.11.20070916-2300

Click on “**Install All**” button.

10. Finally the installer will display the following message:


Install/Update

You will need to restart the Eclipse SDK for the changes to take effect. Would you like to restart now?

Now the Eclipse-Maven plugin is ready for use.

3. DataNucleus –JDO

DataNucleus and Eclipse

Eclipse provides a powerful development environment for Java systems. DataNucleus provides its own plugin for use within Eclipse, giving access to many features of DataNucleus from the convenience of your development environment.

- Installation
- General Preferences
- Preferences : Enhancer
- Preferences : SchemaTool
- Enable DataNucleus Support
- Generate JDO 2 MetaData
- Generate persistence.xml
- Run the Enhancer
- Run SchemaTool

Plugin Installation

The DataNucleus plugin requires Eclipse 3.1 or above. To obtain and install the DataNucleus Eclipse plugin select

Help -> Software Updates -> Find and Install
On the panel that pops up select

Search for new features to install
Select New Remote Site, and in that new window set the URL as **http://www.datanucleus.org/downloads/eclipse-update/** and the name as DataNucleus. Now select the site it has added "DataNucleus", and click "Finish". This will then find the releases of the DataNucleus plugin. **Select the latest version of the DataNucleus Eclipse plugin** . Eclipse then downloads and installs the plugin. Easy!

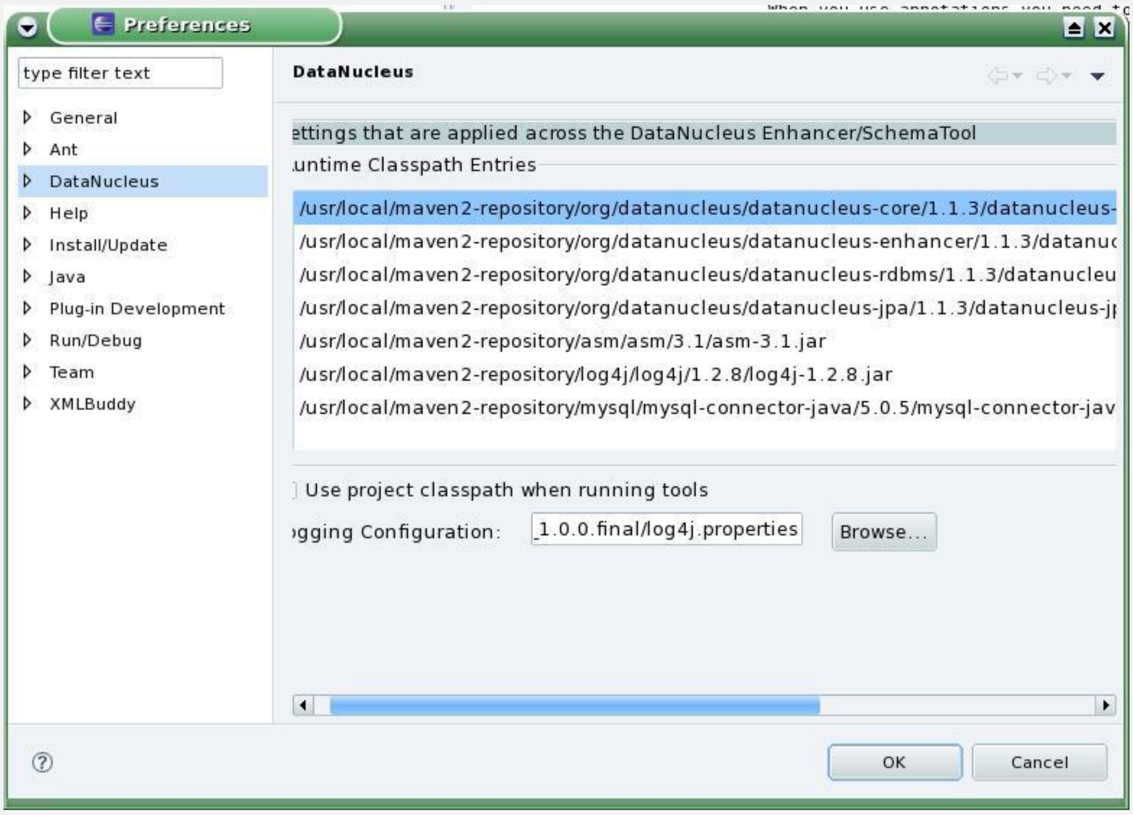
Plugin configuration

The DataNucleus Eclipse plugin allows saving of preferences so that you get nice defaults for all subsequent usage. You can set the preferences at two levels :-

- **Globally for the Plugin** : Go to *Window -> Preferences -> DataNucleus Eclipse Plugin* and see the options below that
- **For a Project** : Go to *{your project} -> Properties -> DataNucleus Eclipse Plugin* and select "Enable project-specific properties"

Plugin configuration - General

Firstly open the main plugin preferences page and configure the libraries needed by DataNucleus. These are in addition to whatever you already have in your projects CLASSPATH, but to run the DataNucleus Enhancer you will require ASM/BCEL (depending which you are using), JDO (but you'll likely have this in your project CLASSPATH), DataNucleus Core/Enhancer/RDBMS jars, as well as LOG4J, and your JDBC driver (if using RDBMS). Below this you can set the location of a configuration file for Log4j to use. This is useful when you want to debug the Enhancer/SchemaTool operations.



DataNucleus

ettings that are applied across the DataNucleus Enhancer/SchemaTool

untime Classpath Entries

```
/usr/local/maven2-repository/org/datanucleus/datanucleus-core/1.1.3/datanucleus-
/usr/local/maven2-repository/org/datanucleus/datanucleus-enhancer/1.1.3/datanuc
/usr/local/maven2-repository/org/datanucleus/datanucleus-rdbms/1.1.3/datanucleu
/usr/local/maven2-repository/org/datanucleus/datanucleus-jpa/1.1.3/datanucleus-jp
/usr/local/maven2-repository/asm/asm/3.1/asm-3.1.jar
/usr/local/maven2-repository/log4j/log4j/1.2.8/log4j-1.2.8.jar
/usr/local/maven2-repository/mysql/mysql-connector-java/5.0.5/mysql-connector-jav
```

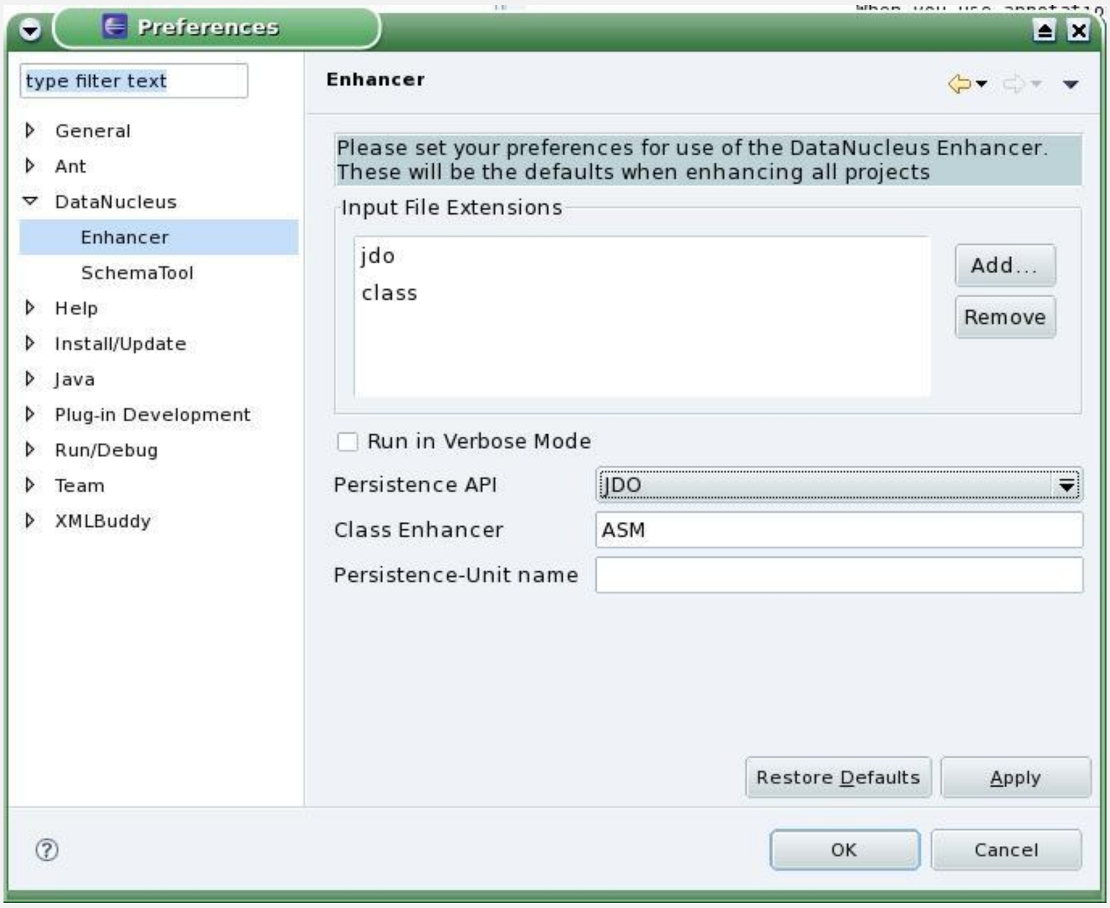
Use project classpath when running tools

ıgging Configuration: _1.0.0.final/log4j.properties

Plugin configuration - Enhancer

Open the "Enhancer" page. You have the following settings

- **Input file extensions** : the enhancer accepts input defining the classes to be enhanced. This is typically performed by passing in the JDO MetaData files. When you use annotations you need to pass in *class* files. So you select the suffices you need
- **Verbose** : selecting this means you get much more output from the enhancer
- **Persistence API** : DataNucleus supports JDO and JPA
- **ClassEnhancer** : DataNucleus provides an enhancer using ASM.



Enhancer

Please set your preferences for use of the DataNucleus Enhancer.
These will be the defaults when enhancing all projects

Input File Extensions
jdo
class

Run in Verbose Mode

Persistence API: JDO

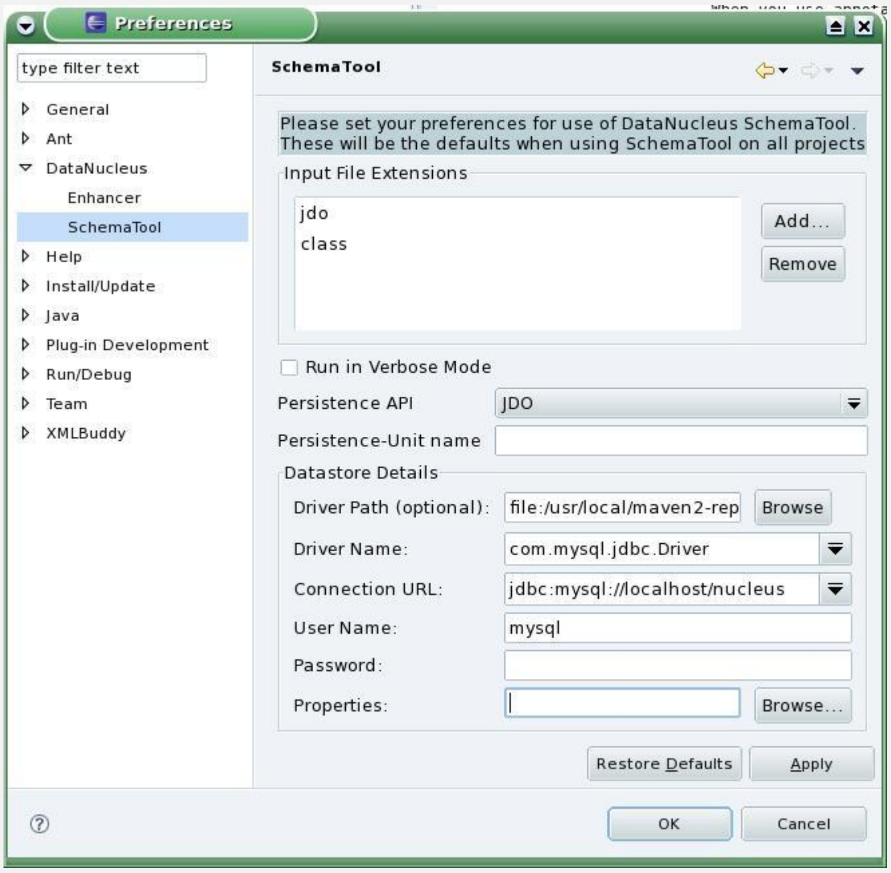
Class Enhancer: ASM

Persistence-Unit name:

Plugin configuration - SchemaTool

Open the "SchemaTool" page. You have the following settings

- **Input file extensions** : SchemaTool accepts input defining the classes to have their schema generated. This is typically performed by passing in the JDO MetaData files. When you use annotations you need to pass in *class* files. So you select the suffices you need
- **Verbose** : selecting this means you get much more output from SchemaTool
- **Persistence API** : DataNucleus supports JDO and JPA
- **Datastore details** : You can either specify the location of a properties file defining the location of your datastore, or you supply the driver name, URL, username and password.



SchemaTool

Please set your preferences for use of DataNucleus SchemaTool.
These will be the defaults when using SchemaTool on all projects

Input File Extensions
jdo
class

☐ Run in Verbose Mode

Persistence API: JDO

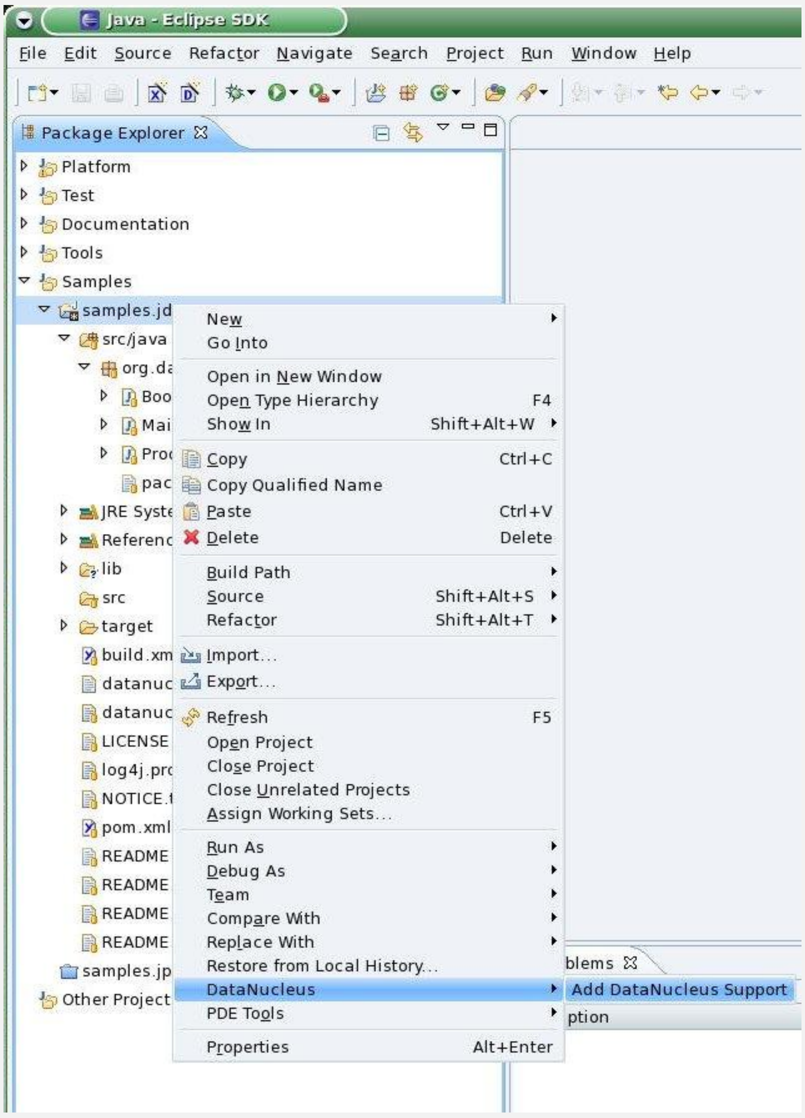
Persistence-Unit name:

Datastore Details
Driver Path (optional): file:/usr/local/maven2-rep
Driver Name: com.mysql.jdbc.Driver
Connection URL: jdbc:mysql://localhost/nucleus
User Name: mysql
Password:
Properties:

Enabling DataNucleus support

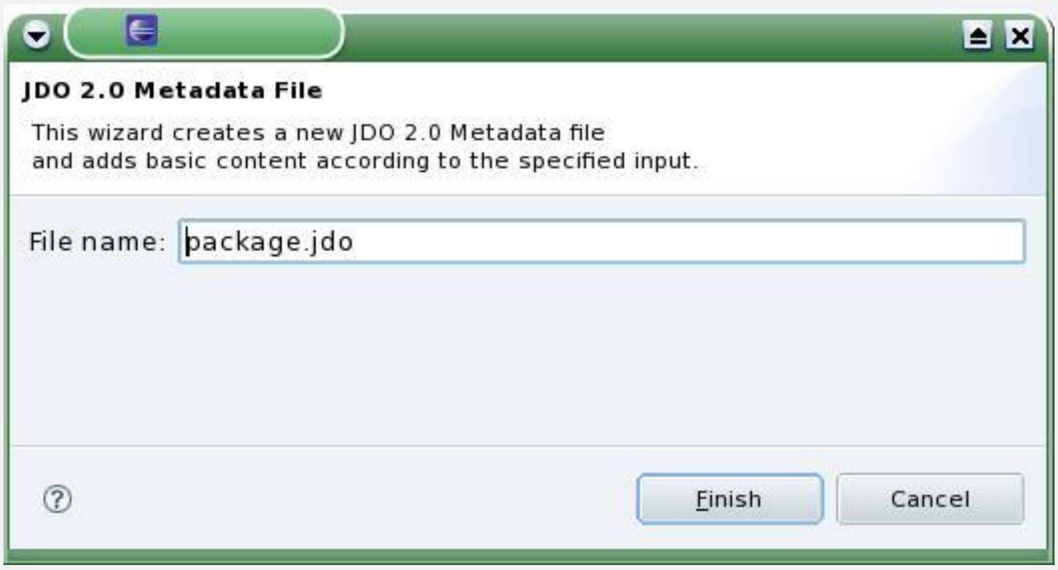
First thing to note is that the DataNucleus plugin is for Eclipse "Java project"s only. After having configured the plugin you can now add DataNucleus support on your projects. Simply right-click on your project and select

DataNucleus->"Add DataNucleus Support"
from the context menu.



Defining JDO2 Metadata

It is standard practice to define the MetaData for your persistable classes in the same package as these classes. You now define your MetaData, by right-click on a package in your project and select "Create JDO 2.0 Metadata File" from DataNucleus context menu. The dialog prompts for the file name to be used and creates a basic Metadata file for all classes in this package, which can now be adapted to your needs. You can also perform same steps as above on a *.java file, which will create the metadata for the selected file only. Please note that the wizard will overwrite existing files without further notice.


JDO 2.0 Metadata File

This wizard creates a new JDO 2.0 Metadata file
and adds basic content according to the specified input.

File name: package.jdo

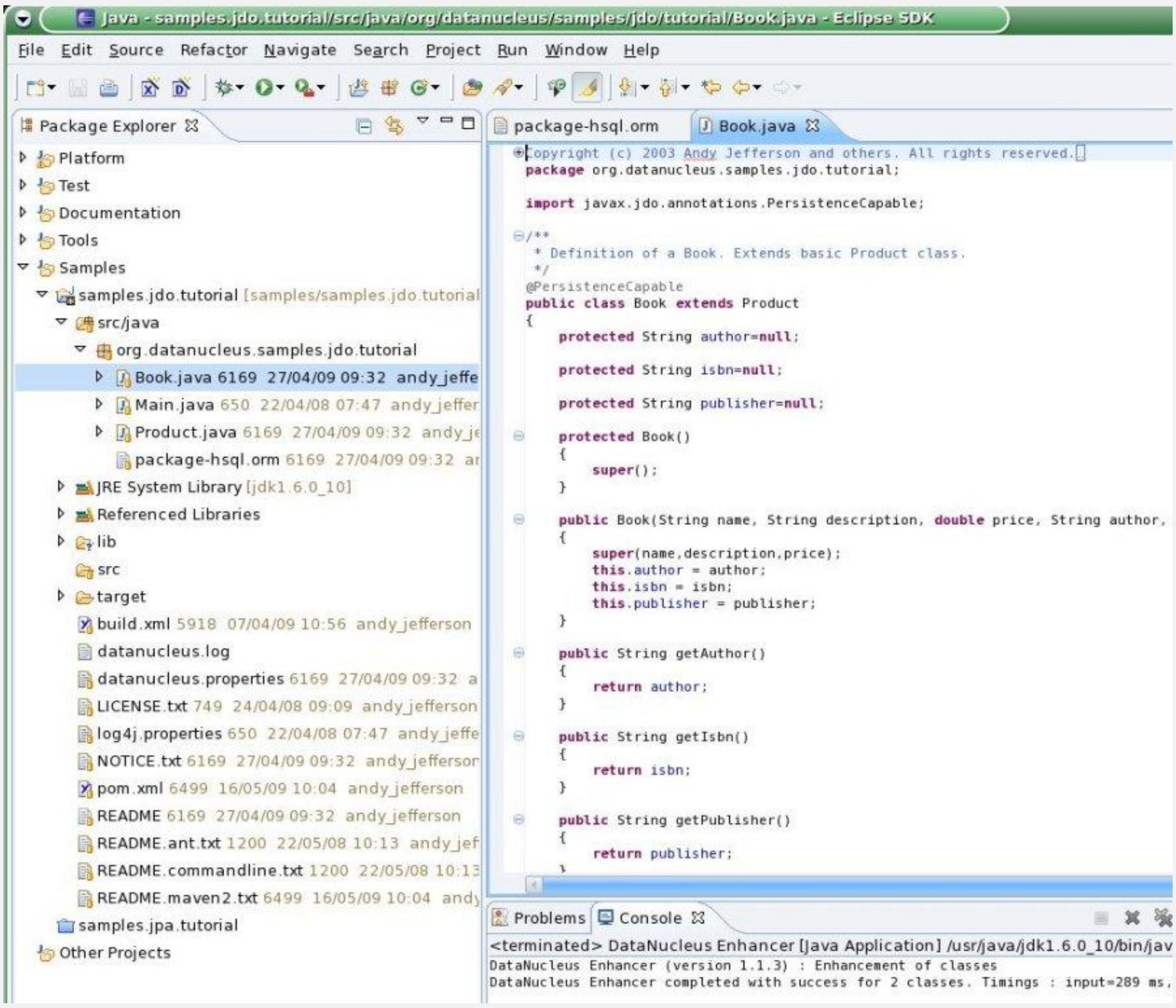
Defining 'persistence.xml'

You can also use the DataNucleus plugin to generate a "persistence.xml" file adding all classes into a single *persistence-unit* . You do this by right-clicking on your project, and selecting the option. The "persistence.xml" is generated under META-INF for the source folder. Please note that the wizard will overwrite existing files without further notice.

Enhancing the classes

The DataNucleus Eclipse plugin allows you to easily byte-code enhance your classes using the DataNucleus enhancer. Right-click on your project and select "Enable Auto-Enhancement" from the DataNucleus context menu. Now that you have the enhancer set up you can enable enhancement of your classes. The DataNucleus Eclipse plugin currently works by enabling/disabling automatic enhancement as a follow on process for the Eclipse build step. This means that when you enable it, every time Eclipse builds your classes it will then enhance the classes defined by the available "jdo" MetaData files. Thereafter every time that you build your classes the JDO enabled ones will be enhanced. Easy! Messages from the enhancement process will be written to the Eclipse Console. **Make sure that you have your**

Java files in a source folder, and that the binary class files are written elsewhere If everything is set-up right, you should see the output below.


Java - samples.jdo.tutorial/src/java/org/datanucleus/samples/jdo/tutorial/Book.java - Eclipse SDK
File Edit Source Refactor Navigate Search Project Run Window Help
Package Explorer
package-hsql.orm
Book.java
Problems
Console

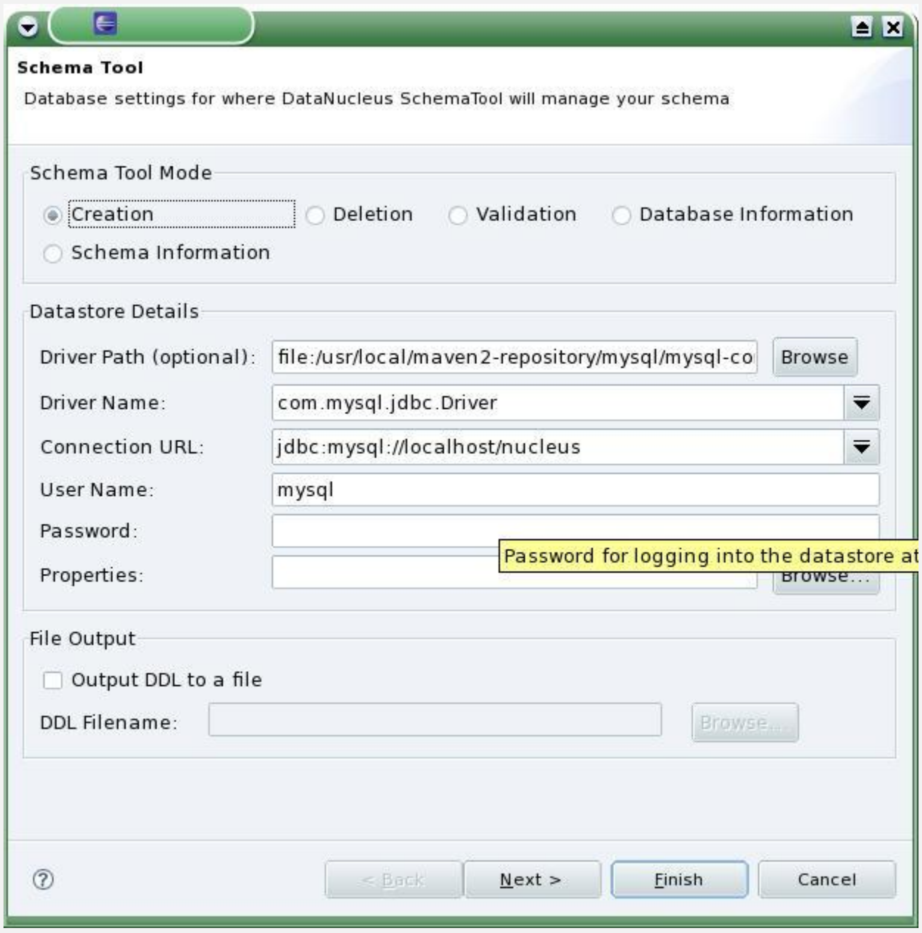
```
Platform
Test
Documentation
Tools
Samples
  samples.jdo.tutorial [samples/samples.jdo.tutorial
    src/java
      org.datanucleus.samples.jdo.tutorial
        Book.java 6169  27/04/09 09:32  andy_jeffe
        Main.java 650  22/04/08 07:47  andy_jeffer
        Product.java 6169  27/04/09 09:32  andy_je
        package-hsql.orm 6169  27/04/09 09:32  ar
    JRE System Library [jdk1.6.0_10]
    Referenced Libraries
    lib
    src
    target
    build.xml 5918  07/04/09 10:56  andy_jefferson
    datanucleus.log
    datanucleus.properties 6169  27/04/09 09:32  a
    LICENSE.txt 749  24/04/08 09:09  andy_jefferson
    log4j.properties 650  22/04/08 07:47  andy_jeffe
    NOTICE.txt 6169  27/04/09 09:32  andy_jefferson
    pom.xml 6499  16/05/09 10:04  andy_jefferson
    README 6169  27/04/09 09:32  andy_jefferson
    README.ant.txt 1200  22/05/08 10:13  andy_jef
    README.commandline.txt 1200  22/05/08 10:13
    README.maven2.txt 6499  16/05/09 10:04  andy
  samples.jpa.tutorial
Other Projects
```

```java
Copyright (c) 2003 Andy Jefferson and others. All rights reserved.
package org.datanucleus.samples.jdo.tutorial;

import javax.jdo.annotations.PersistenceCapable;

/**
 * Definition of a Book. Extends basic Product class.
 */
@PersistenceCapable
public class Book extends Product
{
    protected String author=null;

    protected String isbn=null;

    protected String publisher=null;

    protected Book()
    {
        super();
    }

    public Book(String name, String description, double price, String author,
    {
        super(name,description,price);
        this.author = author;
        this.isbn = isbn;
        this.publisher = publisher;
    }

    public String getAuthor()
    {
        return author;
    }

    public String getIsbn()
    {
        return isbn;
    }

    public String getPublisher()
    {
        return publisher;
    }
```

```
<terminated> DataNucleus Enhancer [Java Application] /usr/java/jdk1.6.0_10/bin/jav
DataNucleus Enhancer (version 1.1.3) : Enhancement of classes
DataNucleus Enhancer completed with success for 2 classes. Timings : input=289 ms,
```

Generating your database schema (RDBMS)

Once your classes have been enhanced you are in a position to create the database schema (assuming you will be using a new schema - omit this step if you already have your schema). Click on the project under "Package Explorer" and under "DataNucleus" there is an option "Run Schema Tool". This brings up a panel to define your database location (URL, login, password etc). You enter these details and the schema will be generated.


Schema Tool

Database settings for where DataNucleus SchemaTool will manage your schema

Schema Tool Mode

[✓ Creation] Deletion Validation Database Information Schema Information

Datastore Details

Driver Path (optional):	file:/usr/local/maven2-repository/mysql/mysql-co
Driver Name:	com.mysql.jdbc.Driver
Connection URL:	jdbc:mysql://localhost/nucleus
User Name:	mysql
Password:	
Properties:	

Password for logging into the datastore at

File Output

☐ Output DDL to a file

DDL Filename:

Messages from the SchemaTool process will be written to the Eclipse Console.

4. GWT Tutorial

This tutorial describes how to develop a Web application with GWT and the Google Plugin for Eclipse. This article assumes basic Eclipse and Java knowledge. The tutorial was developed using JDK 1.6, GWT 2.1 and Eclipse 3.6.

Table of Contents

1. Overview

1.1. Google Web Toolkit

The Google Web Toolkit (GWT) is a toolkit to develop Ajax web application with Java. Once the Java code is finished, the GWT compiler translates the Java code into HTML and Javascript. The compiler creates browser specific HTML and JavaScript to support all the major browsers correctly. GWT supports a standard set of UI widgets, has build in support for the browser back button and a JUnit based test framework.

GWT provides two modes

- Development Mode: allows to debug the Java code of your application directly via the standard Java debugger.
- Web mode: the application is translated into HTML and Javascript code and can be deployed on a webserver.

1.2. Modules, Entry Points and HTML pages

GWT applications are described as so-called modules. A module "modulename" is described by a configuration file "modulename.gwt.xml". Each module can define one or more ***Entry point*** classes. An entry point in GWT is the starting point for a GWT application similar to the main method in a standard Java program.

The module is connected with an HTML page, which is called "host page". The code for a GWT web application executes within this HTML document. The HTML page can define "div" containers to which the GWT application can assign UI components. Alternatively the GWT UI components can be assigned to the body of the HTML page.

1.3. Using CSS

The look and feel of a GWT application can be customized via CSS files. Each widget in GWT can be given a div container by using the method setStyle(String s); We will see later how this is used.

2. Installation of the Google Tools for Eclipse

Google offers a Eclipse plugin that provides both Google App Engine and GWT development capabilities. Install the plugins from http://dl.google.com/eclipse/plugin/3.6 via the Eclipse update manager .

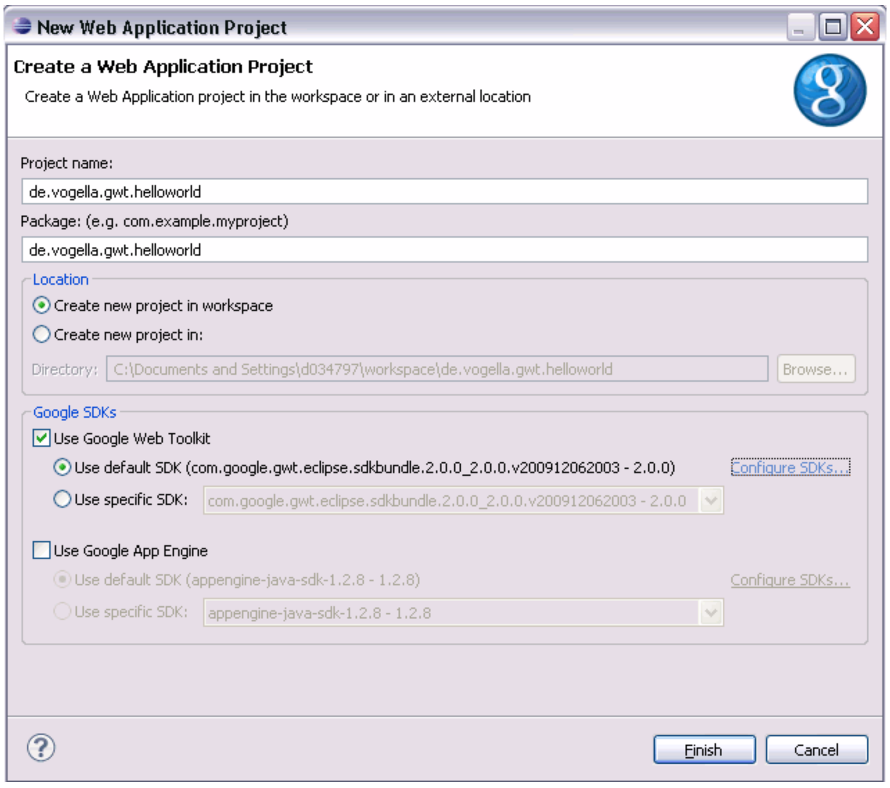
The installation will also setup the GWT and App Engine SDK into your Eclipse preferences. To check this use Window -> Preferences -> Google -> App Engine / Web Toolkit. The SDK are delivered as plugins is included in your Eclipse installation directory under "/plugins/".

3. Create your first GWT application

The following is a description of the usage of the Google plugin to create a new GWT application. plugin.

3.1. Project

Select File > New > Web Application Project. Enter the name "de.vogella.gwt.helloworld" for your project and the java package. Only select "Use Google Web Toolkit", as we will not use the Google App Engine in this tutorial.


New Web Application Project

Create a Web Application Project

Create a Web Application project in the workspace or in an external location

Project name: de.vogella.gwt.helloworld

Package: (e.g. com.example.myproject) de.vogella.gwt.helloworld

Location

[✓ Create new project in workspace]

Create new project in:

Directory: C:\Documents and Settings\d034797\workspace\de.vogella.gwt.helloworld

Google SDKs

[✓ Use Google Web Toolkit]

[✓ Use default SDK (com.google.gwt.eclipse.sdkbundle.2.0.0_2.0.0.v200912062003 - 2.0.0)]

Use specific SDK: com.google.gwt.eclipse.sdkbundle.2.0.0_2.0.0.v200912062003 - 2.0.0

Use Google App Engine

Use default SDK (appengine-java-sdk-1.2.8 - 1.2.8)

Use specific SDK: appengine-java-sdk-1.2.8 - 1.2.8

Unfortunately the Google Plugin does not allow to created a new Web Application Project without creating template files. The template files are nice for the first try but annoying if you want to start from scratch with a new GWT application. Please stare / vote for bug Issue 1547 to get this solved.

Delete all java files under the package "client" and "server".

3.2. GWT entry point

Create a following Java class "HelloGwt" in the "de.vogella.gwt.helloworld.client" package.

```
package de.vogella.gwt.helloworld.client;


import com.google.gwt.core.client.EntryPoint;
import com.google.gwt.event.dom.client.ClickEvent;
import com.google.gwt.event.dom.client.ClickHandler;
import com.google.gwt.user.client.Window;
import com.google.gwt.user.client.ui.Button;
import com.google.gwt.user.client.ui.Label;
import com.google.gwt.user.client.ui.RootPanel;


public class HelloGwt implements EntryPoint {


		@Override
		public void onModuleLoad() {
				Label label = new Label("Hello GWT !!!");
				Button button = new Button("Say something");
				button.addClickHandler(new ClickHandler() {
						@Override
						public void onClick(ClickEvent event) {
								Window.alert("Hello, again");
						}
				});
```

```
                        RootPanel.get().add(label);

                        RootPanel.get().add(button);

            }

}
```

Create the file "De_vogella_gwt_helloworld.gwt.xml" in package "de.vogella.gwt.helloworld" to the following.

```
<?xml version="1.0" encoding="UTF-8"?>

<module rename-to='de_vogella_gwt_helloworld'>

  <!-- Inherit the core Web Toolkit stuff.                        -->

  <inherits name='com.google.gwt.user.User'/>


  <inherits name='com.google.gwt.user.theme.standard.Standard'/>


  <!-- Specify the app entry point class.                         -->

  <entry-point class='de.vogella.gwt.helloworld.client.HelloGwt'/>

</module>
```

This file defines your entry point to your application. In GWT the entry point is similar to the main method in Java.

3.3. HTML page

In the folder war/WEB-INF you should find a folder with the file "De_vogella_gwt_helloworld.html". Change this file to the following.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">

<html>

  <head>

    <meta http-equiv="content-type" content="text/html; charset=UTF-8">

    <link type="text/css" rel="stylesheet" href="De_vogella_gwt_helloworld.css">

    <title>My First GWT applicaton</title>

    <script type="text/javascript" language="javascript"
src="de_vogella_gwt_helloworld/de_vogella_gwt_helloworld.nocache.js"></script>

  </head>



  <body>



    <!-- OPTIONAL: include this if you want history support -->

    <iframe src="javascript:''" id="__gwt_historyFrame" tabIndex='-1'
style="position:absolute;width:0;height:0;border:0"></iframe>

    <h1>My First GWT applicaton</h1>



  </body>

</html>
```

Please note that the HTML files points to the generate JavaScript file in the script tag. If you using a different project name you need to adjust the path.

3.4. web.xml

The Google Plugin created also a web.xml and placed a reference to a servlet (which we deleted). Change the web.xml to the following.

```
<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE web-app

    PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"

    "http://java.sun.com/dtd/web-app_2_3.dtd">

<web-app>



                  <!-- Default page to serve -->

                  <welcome-file-list>

                                    <welcome-file>De_vogella_gwt_helloworld.html</welcome-file>

                  </welcome-file-list>

</web-app>
```

3.5. Modify CSS

Change the CSS of our application. In folder war you find a file "De_vogella_gwt_helloworld.css". Change it to the following.

```
h1 {

  font-size: 2em;

  font-weight: bold;

  color: #777777;

}
```

```
.gwt-Label {

            color: #DF0101;

            font: normal 12px tahoma, arial, helvetica, sans-serif;

            border: 1px solid #99bbe8;

            padding: 14px;

}



.gwt-Button {

            height: 5.7em;

            margin-bottom: 5px;

            padding-bottom: 3px;

            font-size: 12px;

            font-family: arial, sans-serif;

}
```

The css file is referred to by the HTML page.

You can assign a certain style to your GWT UI elements by using the "setStyleName"-Method as shown below.

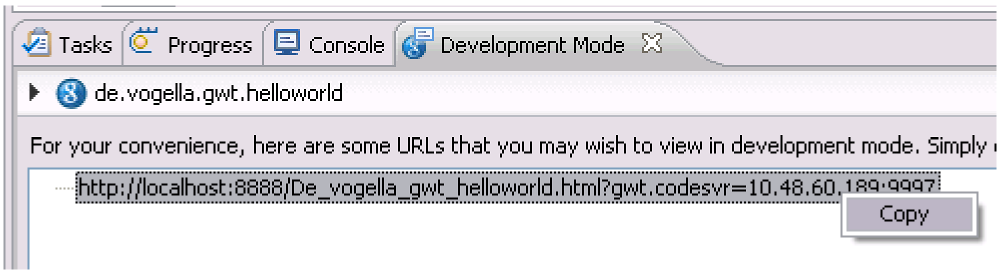
```
package de.vogella.gwt.helloworld.client;


import com.google.gwt.core.client.EntryPoint;
import com.google.gwt.event.dom.client.ClickEvent;
import com.google.gwt.event.dom.client.ClickHandler;
import com.google.gwt.user.client.Window;
import com.google.gwt.user.client.ui.Button;
import com.google.gwt.user.client.ui.Label;
import com.google.gwt.user.client.ui.RootPanel;


public class HelloGwt implements EntryPoint {


	@Override
	public void onModuleLoad() {
		Label label = new Label("Hello GWT !!!");
		label.setStyleName("my_label_for_css"); // Change css to use this
		Button button = new Button("Say something");


		button.addClickHandler(new ClickHandler() {
			@Override
			public void onClick(ClickEvent event) {
				Window.alert("Hello, again");
			}
```

```
                        });



                        RootPanel.get().add(label);

                        RootPanel.get().add(button);

            }

}
```

3.6. Run

To run your application right-click the project and select Run As -> "Web application".

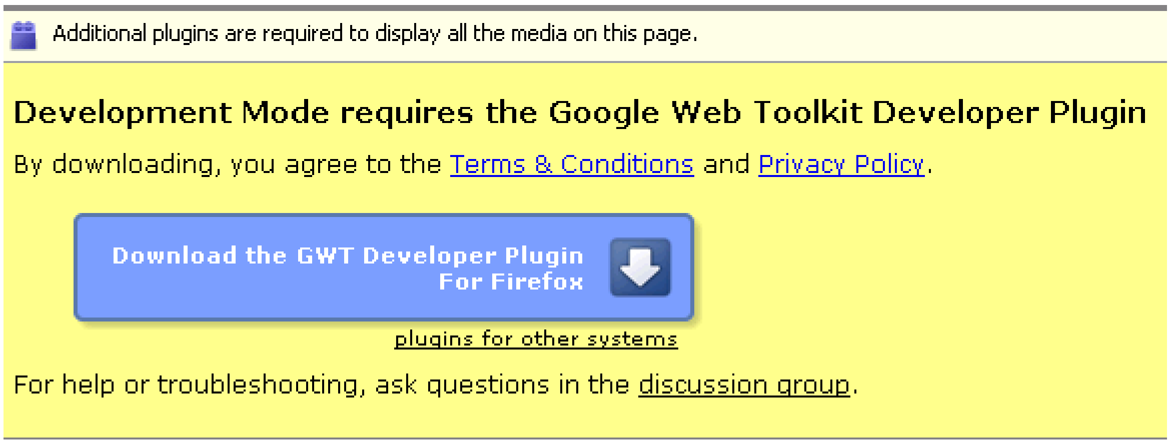
This opens a new view "Development Mode. Copy the url from this view.



For your convenience, here are some URLs that you may wish to view in development mode. Simply

http://localhost:8888/De_vogella_gwt_helloworld.html?gwt.codesvr=10.48.60.189:9997

Paste this url in your browser and install the required plugin (is necessary).


Additional plugins are required to display all the media on this page.

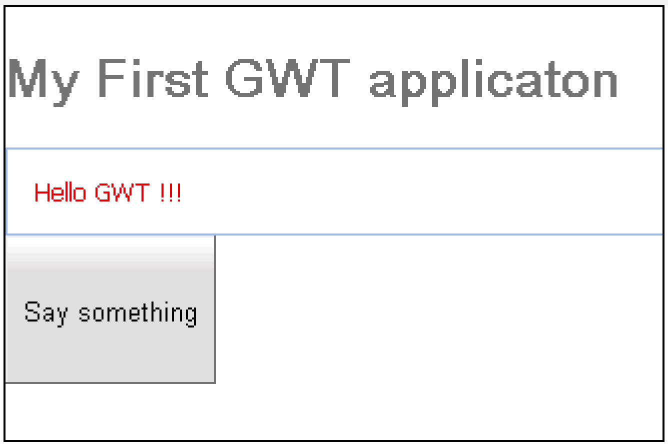
Development Mode requires the Google Web Toolkit Developer Plugin

By downloading, you agree to the Terms & Conditions and Privacy Policy.

Download the GWT Developer Plugin For Firefox
plugins for other systems

For help or troubleshooting, ask questions in the discussion group.

The result should look like this:

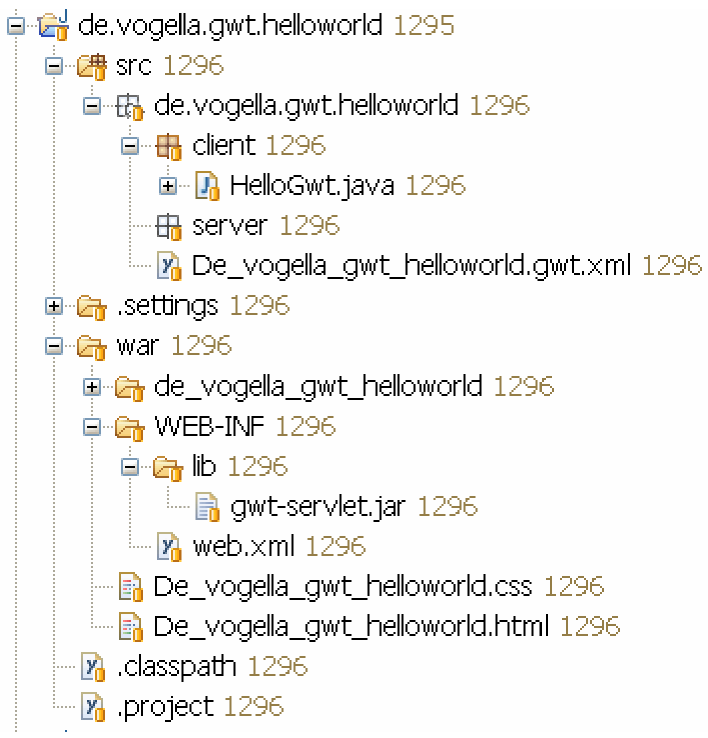


Congratulations! You created your first GWT application.

4. GWT Project Structure

4.1. Overview

If you investigate your example from the last chapter you will see the following project structure.



```
de.vogella.gwt.helloworld 1295
  src 1296
    de.vogella.gwt.helloworld 1296
      client 1296
        HelloGwt.java 1296
      server 1296
      De_vogella_gwt_helloworld.gwt.xml 1296
  .settings 1296
  war 1296
    de_vogella_gwt_helloworld 1296
    WEB-INF 1296
      lib 1296
        gwt-servlet.jar 1296
      web.xml 1296
    De_vogella_gwt_helloworld.css 1296
    De_vogella_gwt_helloworld.html 1296
  .classpath 1296
  .project 1296
```

An GWT application consists out of the following parts

- Module descriptor: XML file which specifies mainly the entry point of the application. It has the name of your module plus .gwt.xml
- war: Under the folder "war" you find the HTML page which contains the GWT application. Additional public resources, e.g. other HTML pages, images or css files should be stored here. Here you find also the standard WEB-INF folder.
- Source code: under your default package your find a package "client" All Java code which is compiled for the client (Webbrowser) must be stored under client. For the service you would use the package "server".

4.2. Module Descriptor

The main purpose of the Module descriptor in *.gwt.xml file is to define your entry-point. It also and defines standard GWT css styles which you are using.

```
<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE module PUBLIC "-//Google Inc.//DTD Google Web Toolkit 1.6.4//EN" "http://google-web-toolkit.googlecode.com/svn/tags/1.6.4/distro-source/core/src/gwt-module.dtd">

<module rename-to='de_vogella_gwt_helloworld'>

  <!-- Inherit the core Web Toolkit stuff.                        -->

  <inherits name='com.google.gwt.user.User'/>



  <inherits name='com.google.gwt.user.theme.standard.Standard'/>



  <!-- Specify the app entry point class.                         -->

  <entry-point class='de.vogella.gwt.helloworld.client.HelloGwt'/>

</module>
```

4.3. Entry point

An entry point in GWT is the starting point for a GWT application, similar to the main method in standard Java program. Every java class which is an entry point must implement the interface "com.google.gwt.core.client.EntryPoint" which defines one method onModuleLoad().

5. Debugging GWT applications

Debugging a GWT application in development mode is easy. In this case you can just debug the Java code. Put a breakpoint into your coding and start the debugger via selecting your project, right-mouse click -> debug as -> Web Application.

You can then use standard Eclipse debugging capabilities. Please see Eclipse Debugging for details.

6. Client and Server Communication (RPC)

6.1. Overview

GWT provides its own remote procedure calls (RPC's) which allow the GWT client to call server-side methods. The implementation of GWT RPC is based on the servlet technology. GWT allows Java objects to be sent directly between the client and the server; which are automatically serialized by the framework. With GWT RPC the communication is almost transparent for the GWT client and always asynchronous so that the client does not block during the communication.

The server-side servlet is usually referred to as a "service" and the a remote procedure call is referred to as "invoking a service." These object can then be used on the client (UI) side.

To create a GWT service you need to define the following:

- An interface which extends RemoteService that lists the service methods.
- An implementation - implements the interface and extends the RemoteServiceServlet.
- Define an asynchronous interface to your service which will be used in the client code

6.2. Create project and domain model

Create a GWT project "de.vogella.gwt.helloserver" with the package "de.vogella.gwt.helloserver". Create the following class which represents the data model . This class implements "Serializable" because GWT requires that all classes that are involved in a server and client communication have to implement the interface "Serializable".

```
package de.vogella.gwt.helloserver.client.model;



import java.io.Serializable;



public class MyUser implements Serializable {

		private static final long serialVersionUID = 1L;



		private String id;

		private String username;

		private String numberOfHits;



		public String getId() {

				return id;

		}



		public void setId(String id) {

				this.id = id;

		}
```

```
public String getUsername() {
		return username;
}

public void setUsername(String username) {
		this.username = username;
}

/**
 * @return the numberOfHits
 */
public String getNumberOfHits() {
		return numberOfHits;
}

/**
 * @param numberOfHits
 *            the numberOfHits to set
 */
public void setNumberOfHits(String numberOfHits) {
		this.numberOfHits = numberOfHits;
}
```

```
            //

}
```

6.3. Interface

The client server communication is based on an interface which defines the possible communication methods.

Create the following interface. By extending the RemoteService interface the GWT compiler will understand that this interface defines an RPC interface. The annotation defines the URL for the service. This must match the entry we will later do in "web.xml".

```
package de.vogella.gwt.helloserver.client.service;



import java.util.List;



import com.google.gwt.user.client.rpc.RemoteService;

import com.google.gwt.user.client.rpc.RemoteServiceRelativePath;



import de.vogella.gwt.helloserver.client.model.MyUser;



//

@RemoteServiceRelativePath("userService")

public interface MyUserService extends RemoteService {
        List<MyUser> getUserList();

        MyUser getUser(String id);

}
```

This interface needs to be available on the client side, therefore it must be placed in the client package.

GWT uses asynchronous communication therefore you also need to create the asynchronous version of this interface. The name of this interface must be the interface name concatenated with "Async".

```
package de.vogella.gwt.helloserver.client.service;



import java.util.List;



import com.google.gwt.user.client.rpc.AsyncCallback;



import de.vogella.gwt.helloserver.client.model.MyUser;



public interface MyUserServiceAsync {

        void getUserList(AsyncCallback<List<MyUser>> callback);

        void getUser(String id, AsyncCallback<MyUser> callback);

}
```

The implementation of this asynchronous interface will be automatically created by the GWT compiler.

6.4. Create the server

For the server implementation create a package "de.vogella.gwt.helloworld.server". Create the following class which extends RemoteServiceServlet.

```
package de.vogella.gwt.helloserver.server;



import java.util.ArrayList;

import java.util.List;



import com.google.gwt.user.server.rpc.RemoteServiceServlet;



import de.vogella.gwt.helloserver.client.model.MyUser;

import de.vogella.gwt.helloserver.client.service.MyUserService;



public class MyUserServiceImpl extends RemoteServiceServlet implements

                        MyUserService {



            private static final long serialVersionUID = 1L;



            private List<MyUser> userList = new ArrayList<MyUser>();



            public MyUserServiceImpl() {
                        MyUser user = new MyUser();

                        user.setId("1");

                        user.setUsername("Peter");

                        user.setNumberOfHits("15");

                        userList.add(user);



                        user = new MyUser();

                        user.setId("2");

                        user.setUsername("Hanz");

                        user.setNumberOfHits("25");

                        userList.add(user);

            }



            public MyUser getUser(String id) {



                        for (Object object : userList) {

                                if (((MyUser) object).getId().equals(id))

                                        return ((MyUser) object);

                        }

                        return null;

            }



            public List<MyUser> getUserList() {

                        return userList;
```

```
            }

}
```

6.5. Define servlet in web.xml

To make GWT aware of this service change the web.xml file in war/WEB-INF to the following.:

```
<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE web-app

    PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"

    "http://java.sun.com/dtd/web-app_2_3.dtd">



<web-app>



  <!-- Default page to serve -->

  <welcome-file-list>

    <welcome-file>De_vogella_gwt_helloserver.html</welcome-file>

  </welcome-file-list>



  <!-- Servlets -->

  <servlet>

    <servlet-name>userServlet</servlet-name>

    <servlet-class>de.vogella.gwt.helloserver.server.MyUserServiceImpl</servlet-class>
  </servlet>



  <servlet-mapping>

    <servlet-name>userServlet</servlet-name>

    <url-pattern>/de_vogella_gwt_helloserver/userService</url-pattern>

  </servlet-mapping>



</web-app>
```

6.6. UI component - Table

Create the following two classes which will be used to display the data in a table. The implementation of "MyTable" is based on FlexTable. FlexTable allows the building of tables in GWT.

```
package de.vogella.gwt.helloserver.client.table;



import java.util.ArrayList;

import java.util.List;



import de.vogella.gwt.helloserver.client.model.MyUser;



public class DataSource {
		private final List<MyUser> users;

		private List<String> header;



		public DataSource(List<MyUser> users) {

			header = new ArrayList<String>();

			header.add("Id");

			header.add("Name");

			header.add("Number of Hits");

			this.users = users;

		}



		public List<MyUser> getUsers() {

			return users;

		}



		public List<String> getTableHeader() {

			return header;

		}



}






package de.vogella.gwt.helloserver.client.table;
```

```
import java.util.List;

import com.google.gwt.user.client.ui.FlexTable;

import de.vogella.gwt.helloserver.client.model.MyUser;

public class MyTable extends FlexTable {
        DataSource input;

        public MyTable(DataSource input) {
                super();
                this.setCellPadding(1);
                this.setCellSpacing(0);
                this.setWidth("100%");
                this.setInput(input);
        }

        public void setInput(DataSource input) {
                for (int i = this.getRowCount(); i > 0; i--) {
                        this.removeRow(0);
                }
                if (input == null) {
                        return;
                }


                int row = 0;

                List<String> headers = input.getTableHeader();

                if (headers != null) {

                        int i = 0;

                        for (String string : headers) {

                                this.setText(row, i, string);

                                i++;

                        }

                        row++;

                }

                // Make the table header look nicer

                this.getRowFormatter().addStyleName(0, "tableHeader");


                List<MyUser> rows = input.getUsers();

                int i = 1;

                for (MyUser myUser : rows) {

                        this.setText(i, 0, myUser.getId());

                        this.setText(i, 1, myUser.getUsername());

                        this.setText(i, 2, myUser.getNumberOfHits());

                        i++;

                }

                this.input = input;
```

```
                }

}
```

6.7. Callback

To receive a callback a class needs to be implemented which can then react on a failure or success in the communication. The interface AsyncCallback defines these two methods: "OnSuccess" and "OnFailure".

Keep in mind that the server communication is asynchronous. If you call your server your GWT application will continue and at some undefined point in time it will receive the data from the server. Make sure that you do not assume in your coding that the call to the server is finished.

Create the following class.

```
package de.vogella.gwt.helloserver.client.service;



import java.util.List;



import com.google.gwt.user.client.Window;

import com.google.gwt.user.client.rpc.AsyncCallback;



import de.vogella.gwt.helloserver.client.model.MyUser;

import de.vogella.gwt.helloserver.client.table.DataSource;

import de.vogella.gwt.helloserver.client.table.MyTable;
```

```
/**
 * Class which handles the asynchronous callback from the server
 * <p>
 * Need to react on server communication failure and success
 *
 * @author Lars Vogel
 *
 */
public class MyUserCallback implements AsyncCallback<List<MyUser>> {

	private MyTable table;

	public MyUserCallback(MyTable table) {
		this.table = table;
	}

	public void onFailure(Throwable caught) {
		Window.alert(caught.getMessage());
	}

	public void onSuccess(List<MyUser> result) {
		List<MyUser> users = result;
		DataSource datasource = new DataSource(users);
```

```
				table.setInput(datasource);

				for (MyUser user : users) {

						System.out.println(user.getUsername());

				}

		}



}
```

This will simple print the received results to the command line.

6.8. Create your entry point

Create the following class.

```
package de.vogella.gwt.helloserver.client.entrypoint;



import com.google.gwt.core.client.EntryPoint;

import com.google.gwt.core.client.GWT;

import com.google.gwt.event.dom.client.ClickEvent;

import com.google.gwt.event.dom.client.ClickHandler;

import com.google.gwt.user.client.rpc.ServiceDefTarget;

import com.google.gwt.user.client.ui.Button;

import com.google.gwt.user.client.ui.DialogBox;

import com.google.gwt.user.client.ui.RootPanel;
import com.google.gwt.user.client.ui.VerticalPanel;


import de.vogella.gwt.helloserver.client.service.MyUserCallback;

import de.vogella.gwt.helloserver.client.service.MyUserService;

import de.vogella.gwt.helloserver.client.service.MyUserServiceAsync;

import de.vogella.gwt.helloserver.client.table.MyTable;


/**

 * Entry point classes define <code>onModuleLoad()</code>.

 */

public class HelloServer implements EntryPoint {


	private MyTable table;


	/**

	 * This is the entry point method.

	 */

	public void onModuleLoad() {

		table = new MyTable(null);


		Button button = new Button("Click me");


		// We can add style names

		button.addStyleName("pc-template-btn");
```

```
// or we can set an id on a specific element for styling


VerticalPanel vPanel = new VerticalPanel();

vPanel.setWidth("100%");

vPanel.setHorizontalAlignment(VerticalPanel.ALIGN_CENTER);

vPanel.add(button);


vPanel.add(table);


// Add table and button to the RootPanel

RootPanel.get().add(vPanel);


// Create the dialog box

final DialogBox dialogBox = new DialogBox();

dialogBox.setText("Welcome to GWT Server Communication!");

dialogBox.setAnimationEnabled(true);

Button closeButton = new Button("close");

VerticalPanel dialogVPanel = new VerticalPanel();

dialogVPanel.setWidth("100%");

dialogVPanel.setHorizontalAlignment(VerticalPanel.ALIGN_CENTER);

dialogVPanel.add(closeButton);


closeButton.addClickHandler(new ClickHandler() {

        @Override
```

```
                public void onClick(ClickEvent event) {
                    dialogBox.hide();
                }
            });

            // Set the contents of the Widget
            dialogBox.setWidget(dialogVPanel);

            button.addClickHandler(new ClickHandler() {
                @Override
                public void onClick(ClickEvent event) {
                    MyUserServiceAsync service = (MyUserServiceAsync) GWT
                            .create(MyUserService.class);
                    ServiceDefTarget serviceDef = (ServiceDefTarget) service;
                    serviceDef.setServiceEntryPoint(GWT.getModuleBaseURL()
                            + "userService");
                    MyUserCallback myUserCallback = new MyUserCallback(table);
                    service.getUserList(myUserCallback);
                }
            });

    }
}
```

Define your entry point in "De_vogella_gwt_helloserver.gwt.xml".

```
<?xml version="1.0" encoding="UTF-8"?>



<!DOCTYPE module PUBLIC "-//Google Inc.//DTD Google Web Toolkit 1.6.4//EN" "http://google-web-toolkit.googlecode.com/svn/tags/1.6.4/distro-source/core/src/gwt-module.dtd">



<module rename-to='de_vogella_gwt_helloserver'>

  <inherits name='com.google.gwt.user.User'/>

  <inherits name='com.google.gwt.user.theme.standard.Standard'/>



  <!-- Specify the app entry point class.                         -->

  <entry-point class='de.vogella.gwt.helloserver.client.entrypoint.HelloServer'/>



</module>
```

Change your "De_vogella_gwt_helloserver.html" page to the following.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">



<html>

  <head>
```

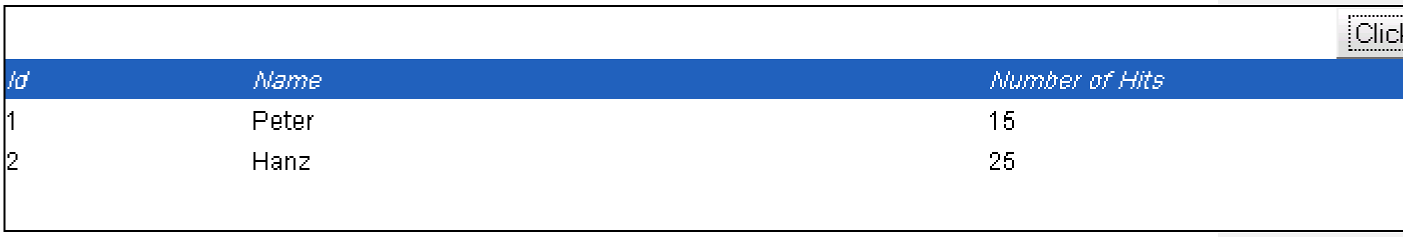
```
    <meta http-equiv="content-type" content="text/html; charset=UTF-8">

    <link type="text/css" rel="stylesheet" href="De_vogella_gwt_helloserver.css">

    <title>Web Application Starter Project</title>



    <script type="text/javascript" language="javascript"
src="de_vogella_gwt_helloserver/de_vogella_gwt_helloserver.nocache.js"></script>

  </head>



  <body>

</body>

</html>
```

Also change the "De_vogella_gwt_helloserver.css" to the following to make the table header look nicer.

```
body {

  padding: 10px;

}



/* stock list header row */

.tableHeader {

  background-color: #2062B8;

  color: white;
  font-style: italic;

}
```

6.9. Run

Run your application. If you click the button then a list of users on click should get loaded.



Id	Name	Number of Hits
1	Peter	15
2	Hanz	25

The eclipse console shows the values of the users send to your Google Web application via the service.

7. UIBinder

UIBinder allows to design GWT UI's declarative via XML.

Create a new GWT project "de.vogella.gwt.uibinder" with the package "de.vogella.gwt.uibinder".

To use UIBinder you also need to inherit from "com.google.gwt.uibinder.UiBinder" in your "gwt.xml" file.

```
<?xml version="1.0" encoding="UTF-8"?>

<module rename-to='de_vogella_gwt_uibinder'>

                <inherits name='com.google.gwt.user.User' />

                <inherits name='com.google.gwt.uibinder.UiBinder' />
```

```
          <inherits name='com.google.gwt.user.theme.standard.Standard' />



          <!-- Specify the app entry point class.                    -->

          <entry-point class='de.vogella.gwt.uibinder.client.De_vogella_gwt_uibinder' />



          <!-- Specify the paths for translatable code               -->

          <source path='client' />



</module>
```

In your client package create the file "HelloWidgetWorld.ui.xml".

```

<ui:UiBinder xmlns:ui='urn:ui:com.google.gwt.uibinder'

          xmlns:g='urn:import:com.google.gwt.user.client.ui'>

          <g:VerticalPanel>

                    <g:HorizontalPanel>

                              <g:Label>Label 2</g:Label>

                              <g:TextBox ui:field='text1'>Some Text</g:TextBox>

                    </g:HorizontalPanel>



                    <g:HorizontalPanel>

                              <g:Label>Label 2 </g:Label>

                              <g:TextBox ui:field='text2'>More Text</g:TextBox>
                    </g:HorizontalPanel>



                    <g:ListBox ui:field='listBox' visibleItemCount='1' />



                    <g:Button ui:field='submit'> Submit </g:Button>

          </g:VerticalPanel>

</ui:UiBinder>
```

Create the following class which uses the xml file for building actions. By convention the name of the xml must equal to the class name + "ui.xml".

```
package de.vogella.gwt.uibinder.client;



import com.google.gwt.core.client.GWT;

import com.google.gwt.event.dom.client.ClickEvent;

import com.google.gwt.uibinder.client.UiBinder;

import com.google.gwt.uibinder.client.UiField;

import com.google.gwt.uibinder.client.UiHandler;

import com.google.gwt.uibinder.client.UiTemplate;

import com.google.gwt.user.client.Window;

import com.google.gwt.user.client.ui.Button;

import com.google.gwt.user.client.ui.Composite;

import com.google.gwt.user.client.ui.ListBox;

import com.google.gwt.user.client.ui.Widget;
```

```
public class HelloWidgetWorld extends Composite {

        // Annotation not needed as we use the default but this allows to change the path

        @UiTemplate("HelloWidgetWorld.ui.xml")

        interface MyUiBinder extends UiBinder<Widget, HelloWidgetWorld> {

        }



        private static MyUiBinder uiBinder = GWT.create(MyUiBinder.class);



        @UiField

        ListBox listBox;



        @UiField Button submit;





        public HelloWidgetWorld(String... names) {

                // sets listBox

                initWidget(uiBinder.createAndBindUi(this));

                for (String name : names) {

                        listBox.addItem(name);

                }

        }



         @UiHandler("submit")
```

```
            void handleClick(ClickEvent e) {

              Window.alert("Hello, UiBinder");

            }









}
```

Create also in the client package the file "MyHTMLTable.ui.xml" This creates a HTML Panel which allow us to layout the UI controls nicely.

```
<ui:UiBinder xmlns:ui='urn:ui:com.google.gwt.uibinder'

            xmlns:g='urn:import:com.google.gwt.user.client.ui'>

            <g:HTMLPanel>

                        <table>

                                    <tr>

                                                <td>

                                                            <g:Label>Name</g:Label>

                                                </td>

                                                <td>

                                                            <g:TextBox ui:field="name" width="15em" />

                                                </td>
```

```
                </tr>

                <tr>

                    <td>

                        <g:Label>Password</g:Label>

                    </td>

                    <td>

                        <g:TextBox ui:field="password" width="15em" />

                    </td>

                </tr>

                <tr>

                    <td colspan='2'>

                        <g:Button ui:field="logIn" text="login" />

                    </td>

                </tr>

            </table>

        </g:HTMLPanel>

</ui:UiBinder>
```

Create the corresponding "MyHTMLTable.java" file.

```
package de.vogella.gwt.uibinder.client;



import com.google.gwt.core.client.GWT;
import com.google.gwt.uibinder.client.UiBinder;

import com.google.gwt.user.client.ui.Composite;

import com.google.gwt.user.client.ui.Widget;



public class MyHTMLTable extends Composite {

	// Annotation can be used to change the name of the associated xml file

	// @UiTemplate("HelloWidgetWorld.ui.xml")

	interface MyUiBinder extends UiBinder<Widget, MyHTMLTable> {

	}



	private static MyUiBinder uiBinder = GWT.create(MyUiBinder.class);





	public MyHTMLTable(String... names) {

		// sets listBox

		initWidget(uiBinder.createAndBindUi(this));

	}



}
```

In your entry point you can now load the class as an composite.

```
package de.vogella.gwt.uibinder.client;
```

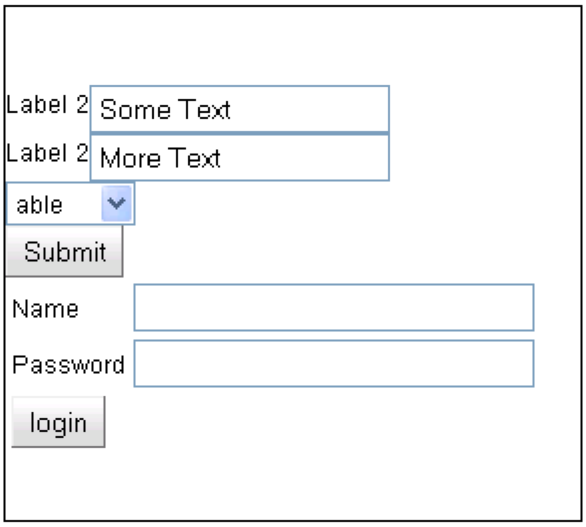
```
import com.google.gwt.core.client.EntryPoint;

import com.google.gwt.user.client.ui.RootPanel;



/**

 * Entry point classes define <code>onModuleLoad()</code>.

 */

public class De_vogella_gwt_uibinder implements EntryPoint {



            public void onModuleLoad() {



                        HelloWidgetWorld helloWorld =

                         new HelloWidgetWorld("able", "baker", "charlie");

    RootPanel.get().add(helloWorld);

    RootPanel.get().add(new MyHTMLTable());

  }



}
```

Change the "web.xml" to the following.

```
<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE web-app
```

```
    PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"

    "http://java.sun.com/dtd/web-app_2_3.dtd">



<web-app>





  <!-- Default page to serve -->

  <welcome-file-list>

    <welcome-file>De_vogella_gwt_uibinder.html</welcome-file>

  </welcome-file-list>



</web-app>
```

The result should look like the following.



8. Using external jars / Java projects in GWT

8.1. Overview

The standard approach in Java is to have separated projects for separate purposes. For example the domain model of the application is usually defined in its own project. One way of making these classes available to GWT is to copy them into the package "client" in your GWT project. This is bad practice as it leads to code duplication (which is inherently evil). This chapter describes how you can make these projects available to the GWT compiler as modules.

GWT need to have access to the source files to compile them into Javascript code. If you add the project or the jar file to your GWT classpath then the Java compiler will not complain if you use the classes from the included project / jar but the GWT compiler will not be able to compile them.

To make the Java files available to the GWT compiler you need to

- Create a gwt.xml file in the Java project / jar file which you want to use - This will instruct the GWT compiler to use the listed classes.
- Use the included library via the inherit definition
- If you are using a jar file you also need to include the source files in the jar

8.2. Create the module project

Create a Java project "de.vogella.gwt.module.model" and package "de.vogella.gwt.module.model". Create the following class.

```
package de.vogella.gwt.module.model;



public class Person {

		private String firstName;



		/**
	 * @return the firstName
	 */
	public String getFirstName() {
		return firstName;
	}

	/**
	 * @param firstName
	 *            the firstName to set
	 */
	public void setFirstName(String firstName) {
		this.firstName = firstName;
	}

}
```

Create in package "de.vogella.gwt.module" the file "model.gwt.xml" with the following content. This will be the module definition for GWT.

```
<module>
  <inherits name='com.google.gwt.user.User'/>
  <source path="model"></source>
</module>
```

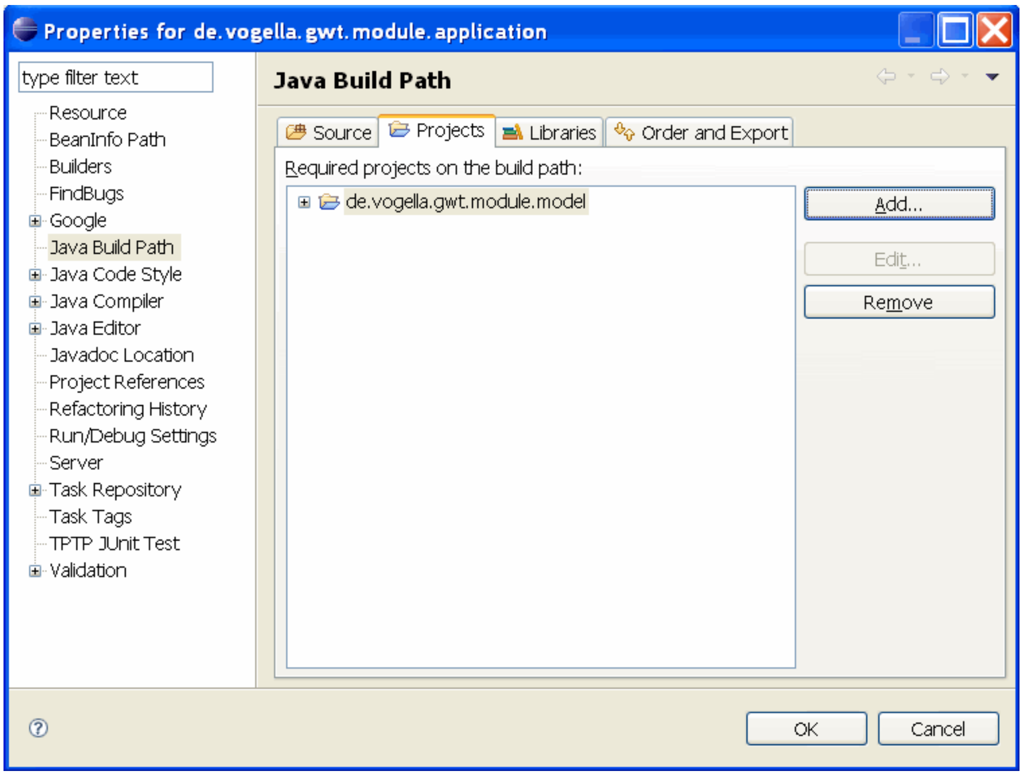
```
}
```

8.3. Use the module in another project

We want to use this model in a GWT project. Create therefore a new GWT project "de.vogella.gwt.module.application" similar to the first example of this article with the following entry point.

```
package de.vogella.gwt.module.application.client;


import com.google.gwt.core.client.EntryPoint;
import com.google.gwt.user.client.ui.Label;
import com.google.gwt.user.client.ui.RootPanel;


import de.vogella.gwt.module.model.Person;


public class ModulTest implements EntryPoint {


	@Override
	public void onModuleLoad() {
		Person p = new Person();
		p.setFirstName("Lars");
		Label label = new Label("Hello " + p.getFirstName());
		RootPanel.get().add(label);
	}
}
```

To make the module project available for the Java compiler, right-click your project, select properties -> Java Build Path and add a dependency to the project "de.vogella.gwt.module.model".



Required projects on the build path:

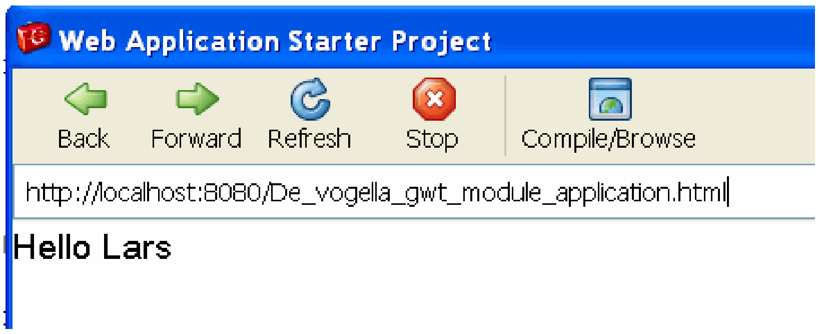
de.vogella.gwt.module.model

Make the Java class of your new module available to the GWT compiler by using "inherits" in your file "De_vogella_gwt_module_application.gwt.xml".

```
<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE module PUBLIC "-//Google Inc.//DTD Google Web Toolkit 1.6.4//EN" "http://google-web-toolkit.googlecode.com/svn/tags/1.6.4/distro-source/core/src/gwt-module.dtd">

<module rename-to='de_vogella_gwt_module_application'>

  <!-- Inherit the core Web Toolkit stuff.                        -->

  <inherits name='com.google.gwt.user.User'/>
  <inherits name='com.google.gwt.user.theme.standard.Standard'/>

  <inherits name='de.vogella.gwt.module.model'/>



  <!-- Other module inherits                                      -->



  <!-- Specify the app entry point class.                         -->

  <entry-point class='de.vogella.gwt.module.application.client.ModulTest'/>

</module>
```

If you make a typo here you will get an error message: [ERROR] Unable to find 'de/vogella/gwt/module/model2.gwt.xml' on your classpath; could be a typo, or maybe you forgot to include a classpath entry for source?

If everything was setup correctly you can run your application.


Web Application Starter Project
Back
Forward
Refresh
Stop
Compile/Browse
http://localhost:8080/De_vogella_gwt_module_application.html

Hello Lars

9. Build your own components

GWT allows you to build your own components based on other widgets. To do this you extends com.google.gwt.user.client.ui.Composite. These customer components can be used as normal GWT components.

Building components is simple, you just offer the API your component should publish and then you call with each widget the initWidget() method.

For example you can create the following component which is a simple header label (I assume you can easily image a more complex example).

```
package mypackage.client;


import com.google.gwt.user.client.ui.Composite;

import com.google.gwt.user.client.ui.Label;


public class Header extends Composite {

	private Label title;


	public Header(String title) {

		this.title = new Label(title);

		initWidget(this.title);

	}


	public void setStyleName(String style) {

		this.title.setStyleName(style);

	}


	public void setTitle(String title) {

		this.title.setText(title);
```

```
                }



}
```

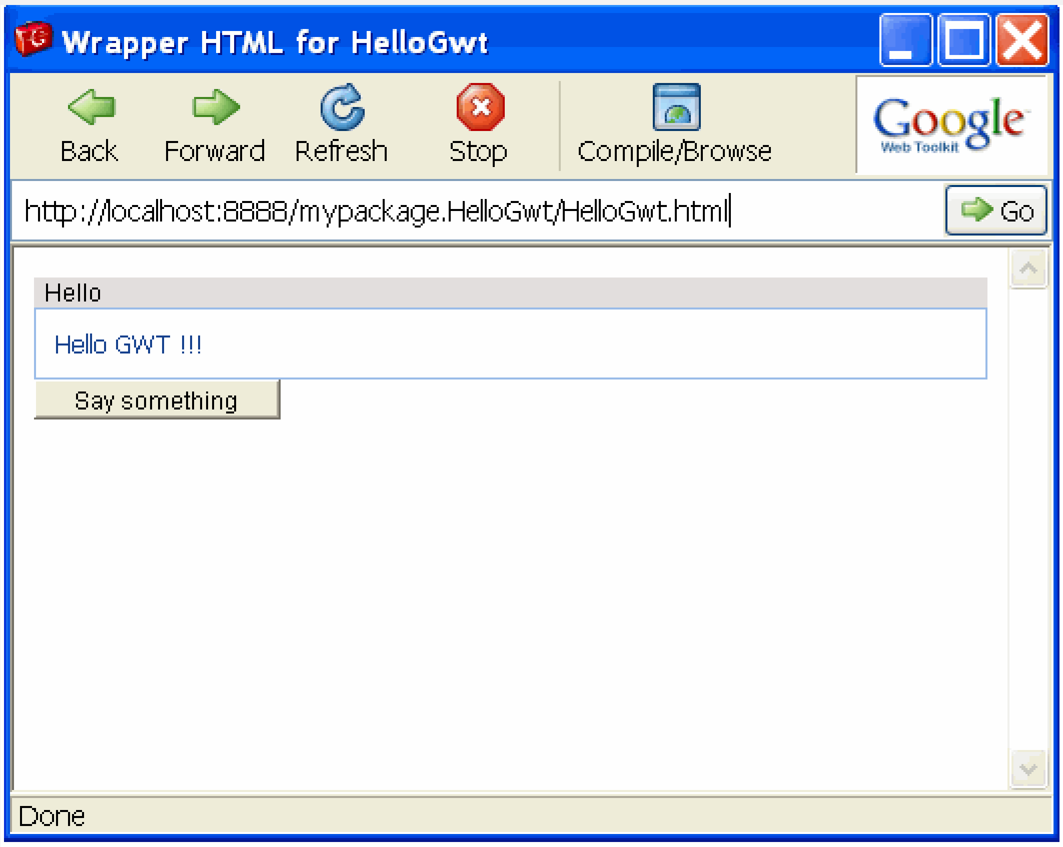
Adjust then the GWT module to use this component.

```
package mypackage.client;



import com.google.gwt.core.client.EntryPoint;

import com.google.gwt.user.client.Window;

import com.google.gwt.user.client.ui.Button;

import com.google.gwt.user.client.ui.ClickListener;

import com.google.gwt.user.client.ui.Label;

import com.google.gwt.user.client.ui.RootPanel;

import com.google.gwt.user.client.ui.Widget;



public class HelloGwt implements EntryPoint {



                public void onModuleLoad() {

                                Header header = new Header("Hello");

                                header.setStyleName("headerpane");

                                Label label = new Label("Hello GWT !!!");

                                label.setStyleName("label");
                                Button button = new Button("Say something");

                                button.addClickListener(new ClickListener() {

                                        public void onClick(Widget sender) {

                                                Window.alert("Hello, again");

                                        }

                                });



                                RootPanel.get().add(header);

                                RootPanel.get().add(label);

                                RootPanel.get().add(button);

                }

}
```

Adjust the css to make the title look nicer.

```
package mypackage.client;



import com.google.gwt.core.client.EntryPoint;

import com.google.gwt.user.client.Window;

import com.google.gwt.user.client.ui.Button;

import com.google.gwt.user.client.ui.ClickListener;

import com.google.gwt.user.client.ui.Label;

import com.google.gwt.user.client.ui.RootPanel;
import com.google.gwt.user.client.ui.Widget;


public class HelloGwt implements EntryPoint {


        public void onModuleLoad() {

                Header header = new Header("Hello");

                header.setStyleName("headerpane");

                Label label = new Label("Hello GWT !!!");

                label.setStyleName("label");

                Button button = new Button("Say something");

                button.addClickListener(new ClickListener() {

                        public void onClick(Widget sender) {

                                Window.alert("Hello, again");

                        }

                });


                RootPanel.get().add(header);

                RootPanel.get().add(label);

                RootPanel.get().add(button);

        }

}
```

The result should look like the following.



5. GWT Designer

This tutorial walks you through the process of creating a simple GWT application and deploying your module using GWT Designer.

Prerequisites:

- Eclipse version 3.3 or higher
- Java 1.4 or higher
- GWT v1.4 or higher
- GWT Designer v5.1 or higher

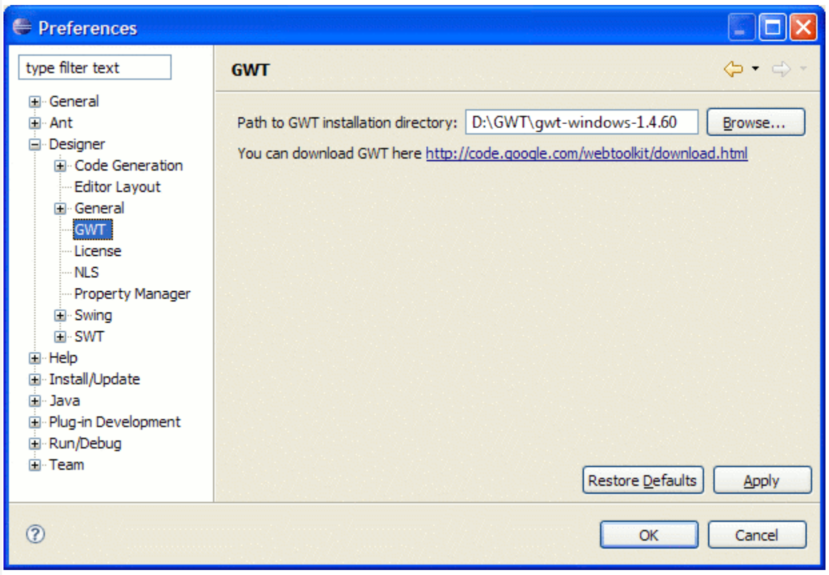
Basic steps:

1. Set the GWT installation directory.
2. Create a GWT Java Project and the LoginManager module.
3. Create the Login Composite.

4. Create and apply CSS styles.
5. Add the Login composite to the LoginManager module.
6. Run Application in Hosted Mode.
7. Build and Deploy.

1. Set the path to the GWT installation directory.

Before anything else, make sure to set the path to the **GWT installation** directory in the **GWT Preferences** has not been set.

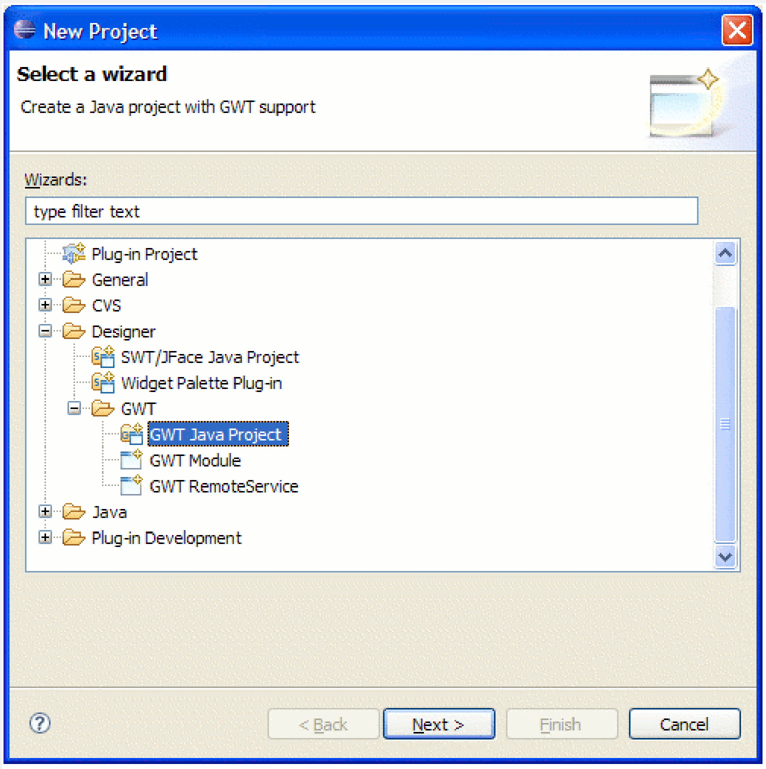


Path to GWT installation directory: D:\GWT\gwt-windows-1.4.60

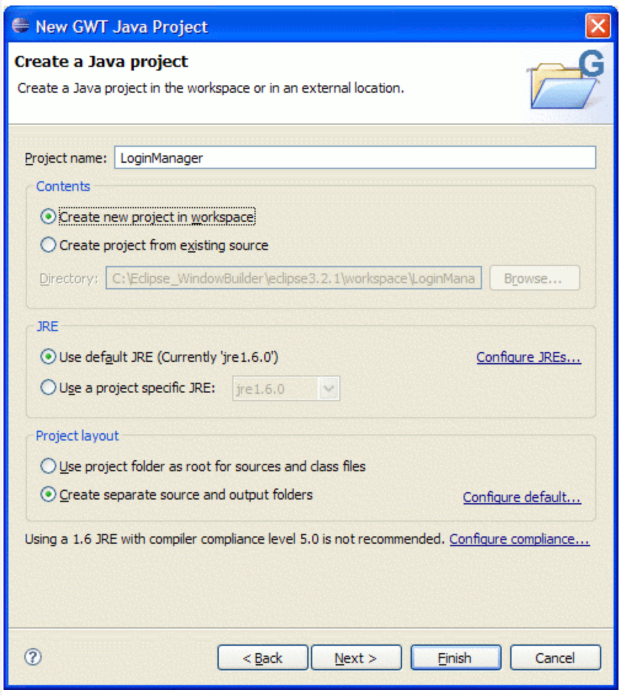
You can download GWT here http://code.google.com/webtoolkit/download.html

2. Create a GWT Java Project and the LoginManager module.

Select **File** > **New** > **Project** from the Eclipse toolbar to open the new project wizard dialog.
Select **Designer** > **GWT** > **GWT Java Project** from the list of wizards.
Click **Next.**



Enter **LoginManager** as the Project name and click **Next.**



Project name: LoginManager

Contents
[✓ Create new project in workspace]
Create project from existing source
Directory: C:\Eclipse_WindowBuilder\eclipse3.2.1\workspace\LoginMana

JRE
[✓ Use default JRE (Currently 'jre1.6.0')]
Use a project specific JRE: jre1.6.0

Project layout
Use project folder as root for sources and class files
[✓ Create separate source and output folders]

Using a 1.6 JRE with compiler compliance level 5.0 is not recommended.

Check the option to **Create a GWT Module.**

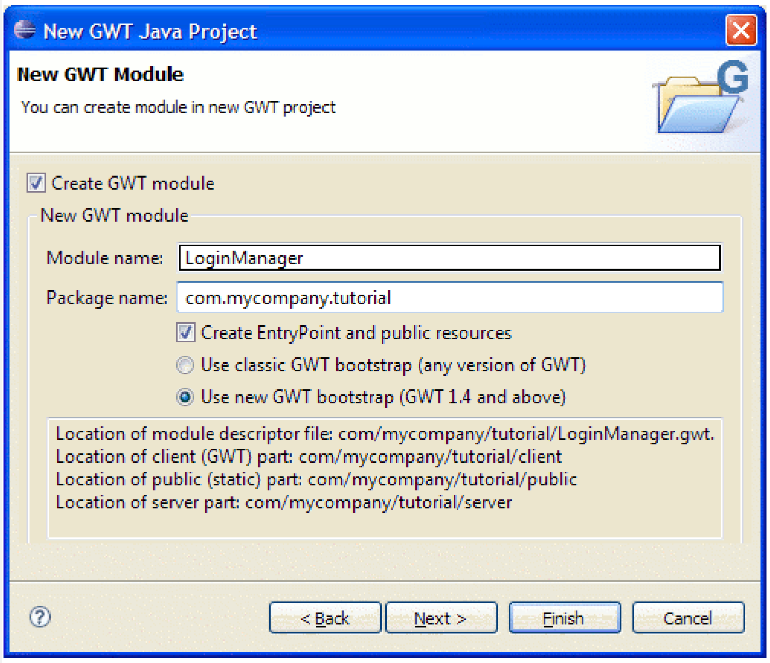
Enter **LoginManager** as the Module name.

Modify the package name to **com.mycompany.tutorial.**

Keep the **Create EntryPoint and public resources** option checked

Select the **Use new GWT bootstrap** option

Click **Finish.**



[✓] Create GWT module

New GWT module

Module name: LoginManager

Package name: com.mycompany.tutorial

[✓] Create EntryPoint and public resources

○ Use classic GWT bootstrap (any version of GWT)

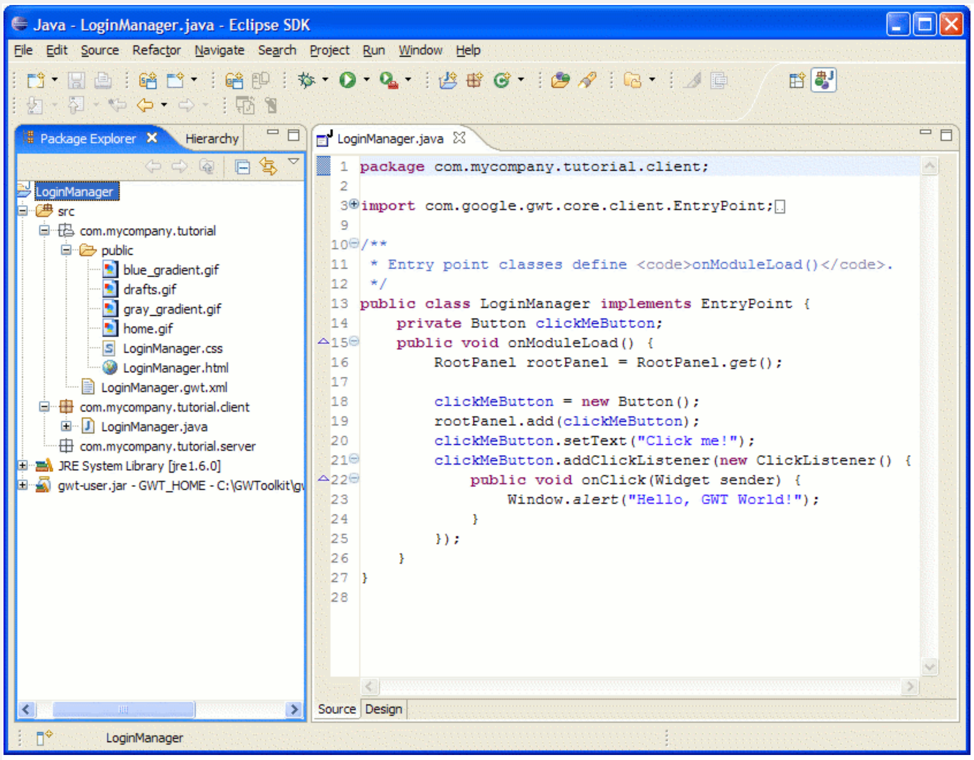
● Use new GWT bootstrap (GWT 1.4 and above)

Location of module descriptor file: com/mycompany/tutorial/LoginManager.gwt.
Location of client (GWT) part: com/mycompany/tutorial/client
Location of public (static) part: com/mycompany/tutorial/public
Location of server part: com/mycompany/tutorial/server

GWT Designer generates the project with all of the necessary configuration and resource files.

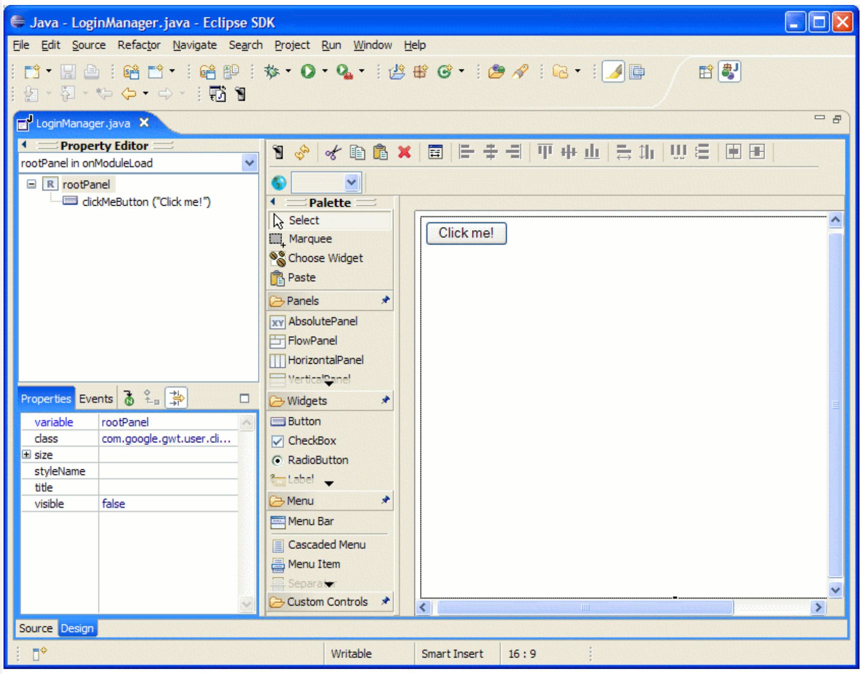
The module **LoginManager.java** is opened in source mode by default.

Click the **Design tab** at the bottom of the source editor.



```java
package com.mycompany.tutorial.client;

import com.google.gwt.core.client.EntryPoint;

/**
 * Entry point classes define <code>onModuleLoad()</code>.
 */
public class LoginManager implements EntryPoint {
    private Button clickMeButton;
    public void onModuleLoad() {
        RootPanel rootPanel = RootPanel.get();

        clickMeButton = new Button();
        rootPanel.add(clickMeButton);
        clickMeButton.setText("Click me!");
        clickMeButton.addClickListener(new ClickListener() {
            public void onClick(Widget sender) {
                Window.alert("Hello, GWT World!");
            }
        });
    }
}
```

This is the LoginManager in Design mode. This is the initial module created by GWT Designer.



rootPanel in onModuleLoad

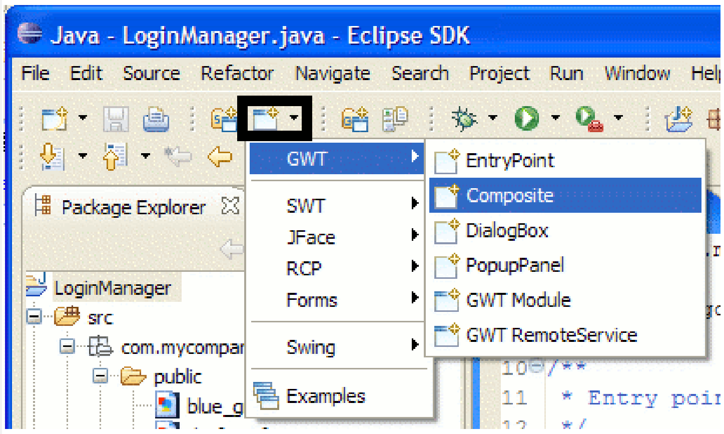
rootPanel
clickMeButton ("Click me!")

variable	rootPanel
class	com.google.gwt.user.cli...
size	
styleName	
title	
visible	false

3. Create the Login Composite.

Create a Composite for the Login application.

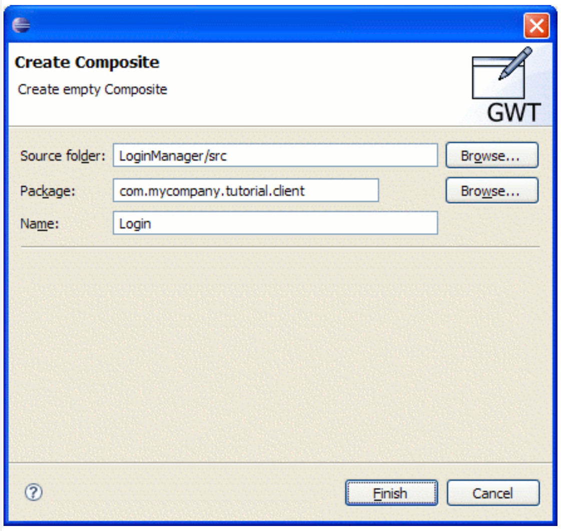
Select the **Designer toolbar button** and select **GWT** > **Composite** from the pull-down menu.



Enter **Login** as the name of the Composite.

Click **Finish**.

Click the Design tab.


Create Composite

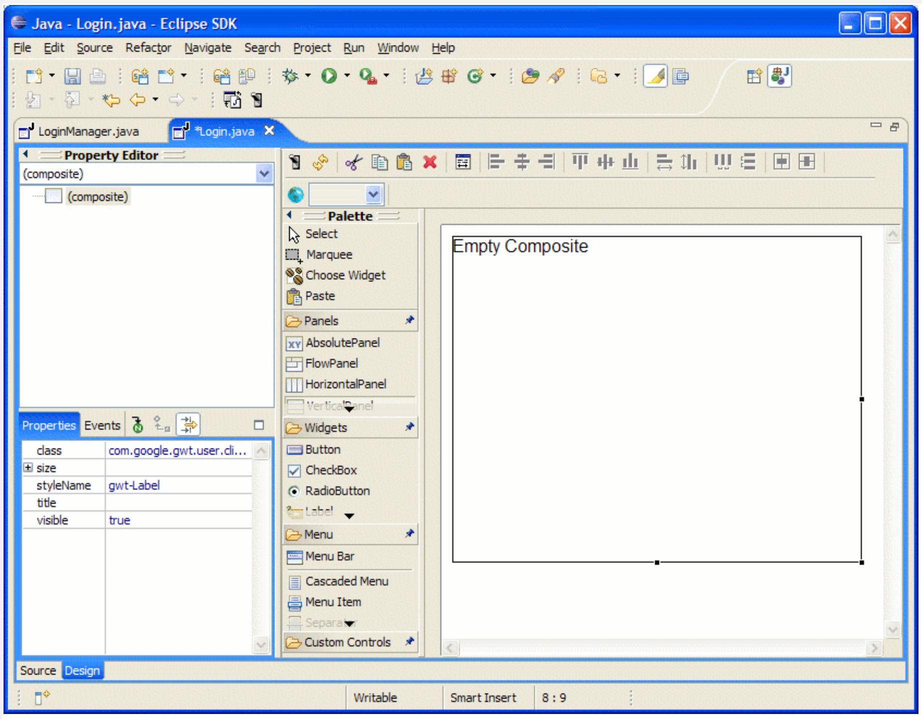
Create empty Composite

Source folder: LoginManager/src

Package: com.mycompany.tutorial.client

Name: Login

You should now see an empty composite.



class	com.google.gwt.user.cli...
size	
styleName	gwt-Label
title	
visible	true

Add a **VerticalPanel** to the Composite.

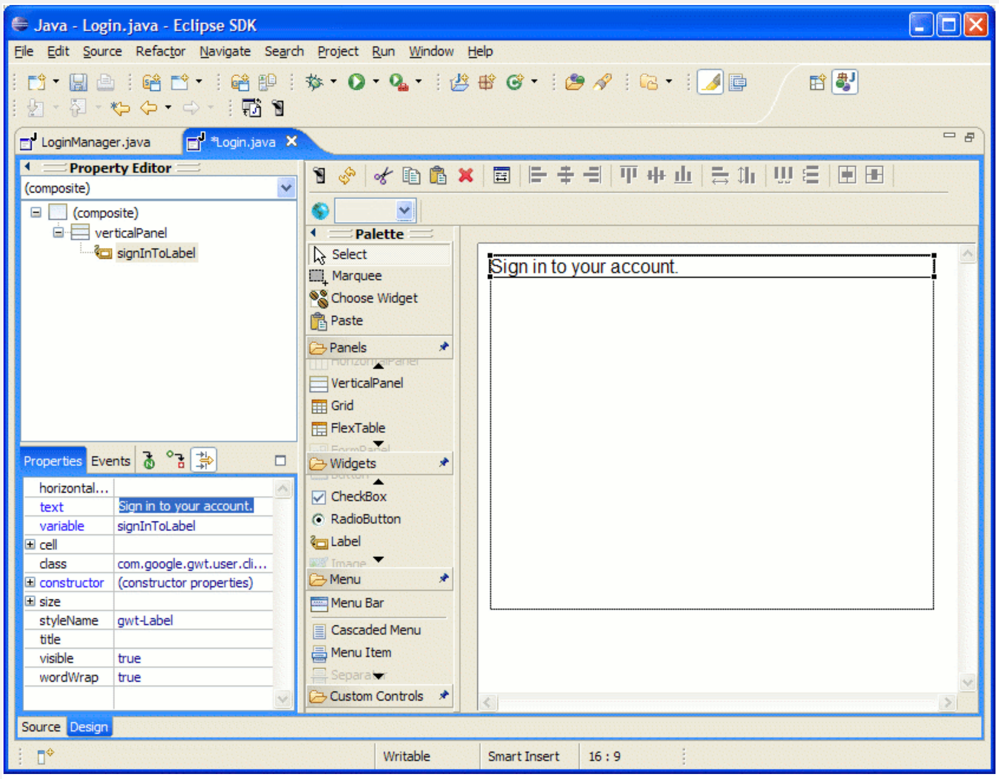
- To do this, select the **VerticalPanel** from the Palette and click on the Login Composite.

  You should now see an Empty VerticalPanel.

Add a label widget to the Composite.

- Select Label from the Palette and add it to the content area.

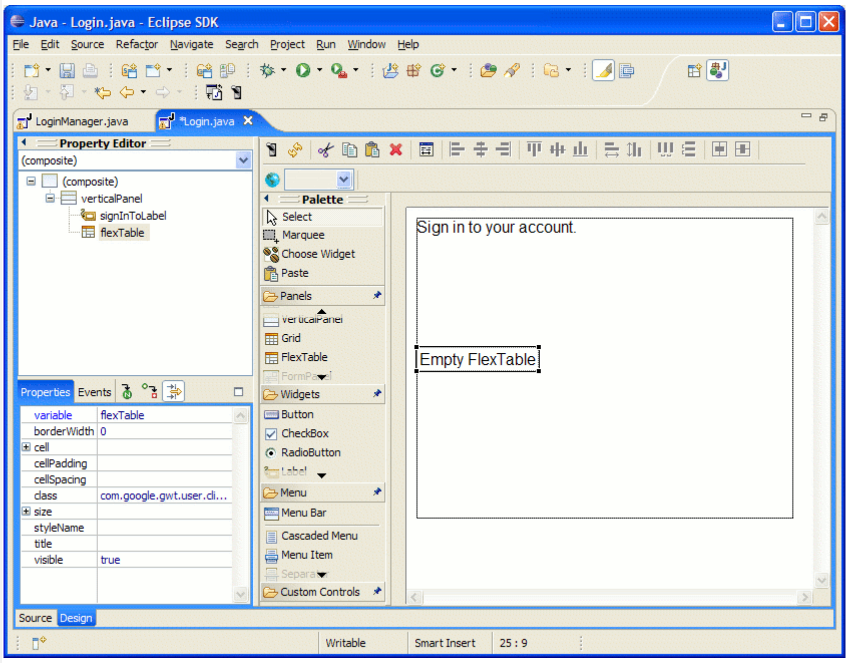
On the Properties tab, set the text property to **Sign in to your account.**



horizontal…	
text	Sign in to your account.
variable	signInToLabel
cell	
class	com.google.gwt.user.cl…
constructor	(constructor properties)
size	
styleName	gwt-Label
title	
visible	true
wordWrap	true

To easily structure the widgets, add a **FlexTable** panel.

- Select **Flextable** under Panels within the Palette and add it to the Composite inside the VerticalPanel.

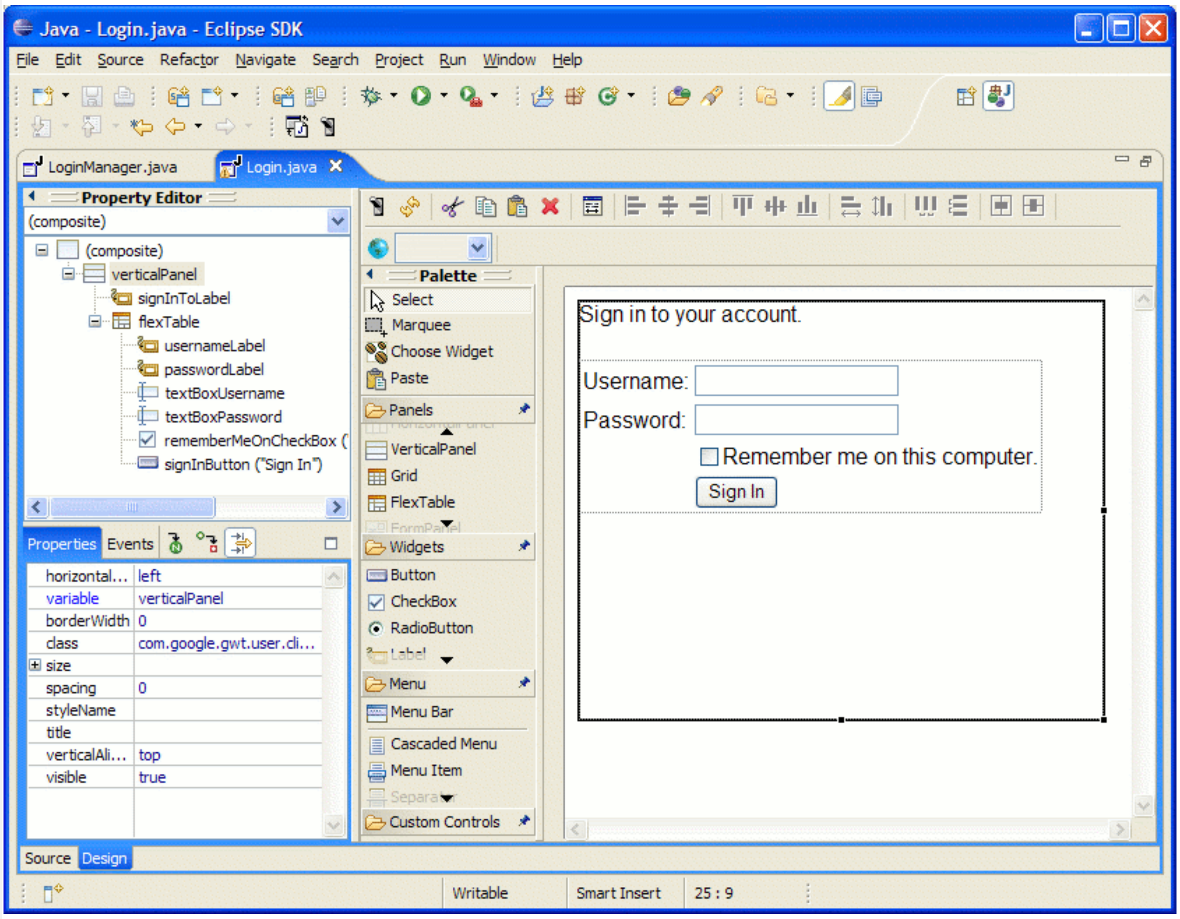


variable	flexTable
borderWidth	0
cell	
cellPadding	
cellSpacing	
class	com.google.gwt.user.cl…
size	
styleName	
title	
visible	true

Add the following widgets and arrange it as seen on the screen shot:

- Add a **Label** and change the text property to **Username:**
- Add a **TextBox** and change the variable property to **textBoxUsername.**
- Add another **Label** and change the text property to **Password:**
- Add another **TextBox** and change the variable property to **textBoxPassword**.
- Add a **CheckBox** and change the text property to **Remember me on this computer**.
- Add a Button below the checkbox and set the text property to **Sign In**.

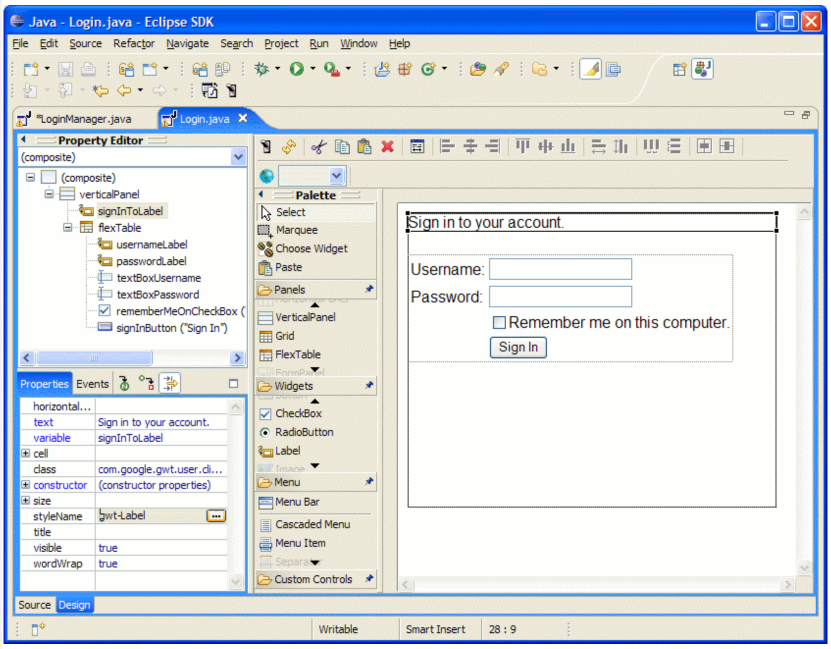
Save your work.



horizontal...	left
variable	verticalPanel
borderWidth	0
class	com.google.gwt.user.cli...
size	
spacing	0
styleName	
title	
verticalAli...	top
visible	true

4. Create and apply CSS styles.

Select one of the labels and click on the ellipsis on the **styleName** property.



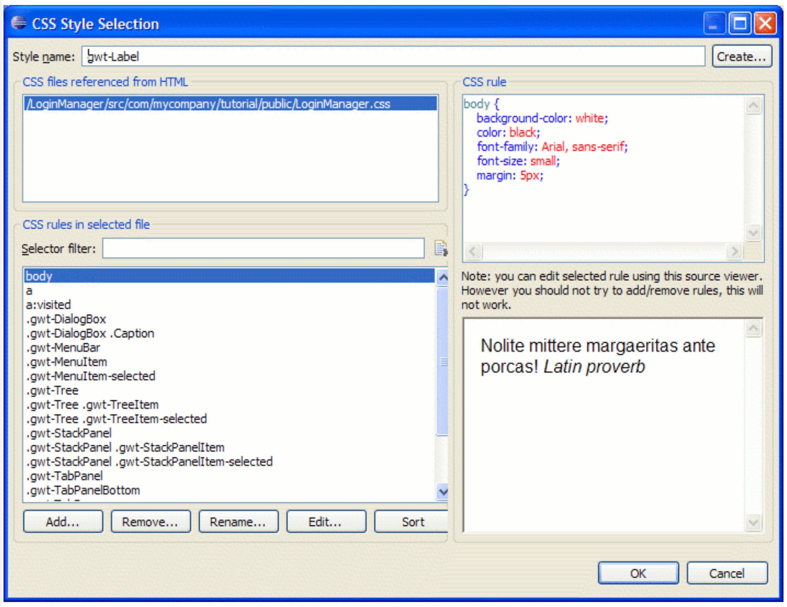
Properties:

horizontal...	
text	Sign in to your account.
variable	signInToLabel
cell	
class	com.google.gwt.user.cli...
constructor	(constructor properties)
size	
styleName	gwt-Label
title	
visible	true
wordWrap	true

That brings up the CSS Style Selection dialog.

Click on the **Add...** button.

On the **New rule dialog**, click **OK.**


CSS Style Selection
Style name: gwt-Label
Create...
CSS files referenced from HTML
/LoginManager/src/com/mycompany/tutorial/public/LoginManager.css
CSS rules in selected file
Selector filter:
CSS rule
Add...
Remove...
Rename...
Edit...
Sort
OK
Cancel

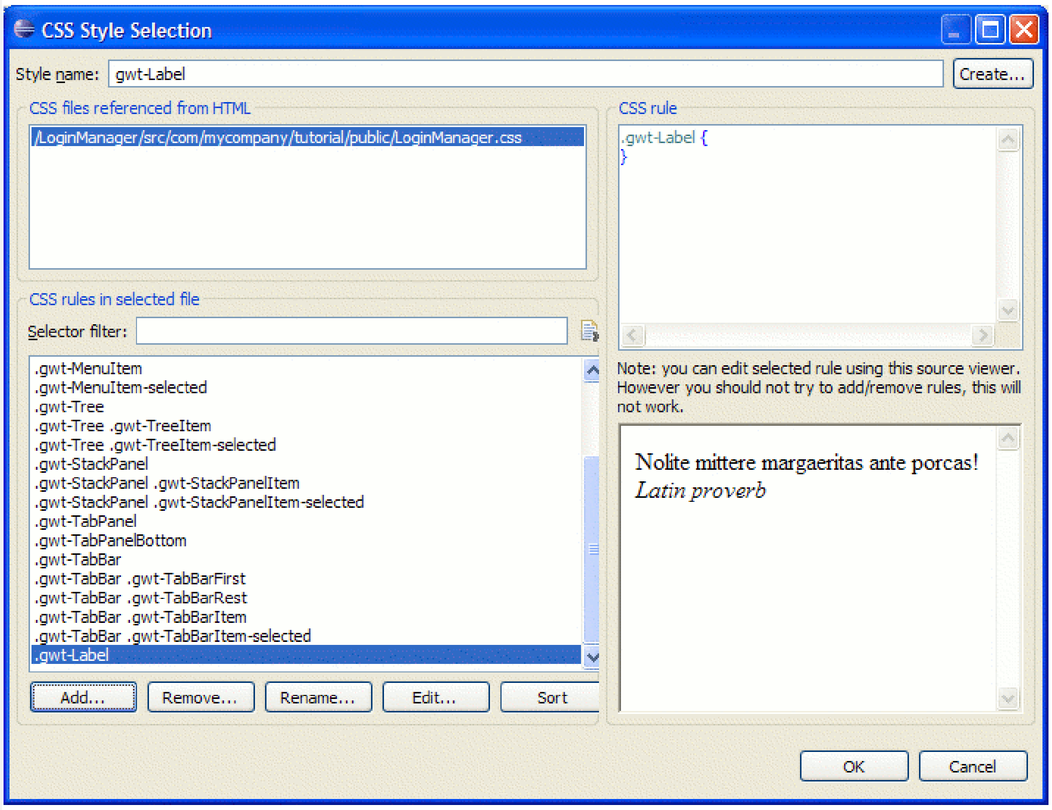
```
body {
  background-color: white;
  color: black;
  font-family: Arial, sans-serif;
  font-size: small;
  margin: 5px;
}
```

Note: you can edit selected rule using this source viewer. However you should not try to add/remove rules, this will not work.

Nolite mittere margaeritas ante porcas! *Latin proverb*

A new css class is added called **.gwt-Label.**

With the .gwt-Label selected, click the **Edit...** button.



Style name: gwt-Label

/LoginManager/src/com/mycompany/tutorial/public/LoginManager.css

```
.gwt-Label {
}
```

Note: you can edit selected rule using this source viewer. However you should not try to add/remove rules, this will not work.

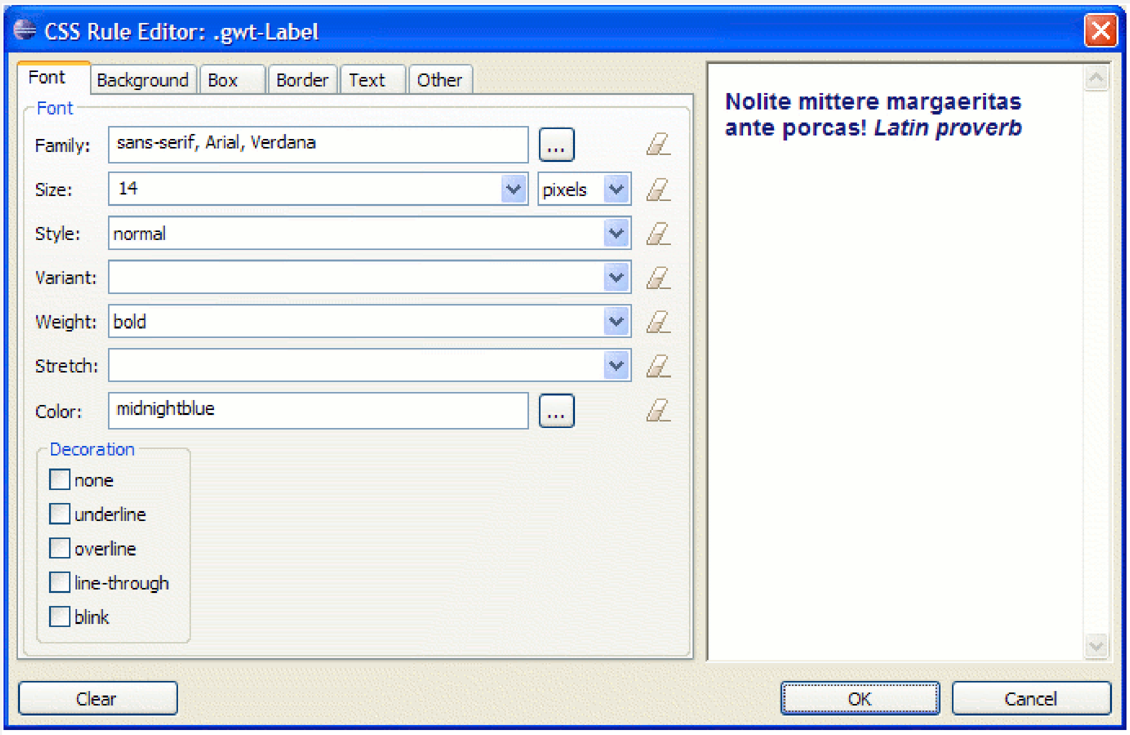
```
.gwt-MenuItem
.gwt-MenuItem-selected
.gwt-Tree
.gwt-Tree .gwt-TreeItem
.gwt-Tree .gwt-TreeItem-selected
.gwt-StackPanel
.gwt-StackPanel .gwt-StackPanelItem
.gwt-StackPanel .gwt-StackPanelItem-selected
.gwt-TabPanel
.gwt-TabPanelBottom
.gwt-TabBar
.gwt-TabBar .gwt-TabBarFirst
.gwt-TabBar .gwt-TabBarRest
.gwt-TabBar .gwt-TabBarItem
.gwt-TabBar .gwt-TabBarItem-selected
.gwt-Label
```

Nolite mittere margaeritas ante porcas!
Latin proverb

The CSS graphical editor dialog comes up.

Feel free to set any attribute you want. When you're done, click OK.

Following the same procedure, create a css style for the checkbox.

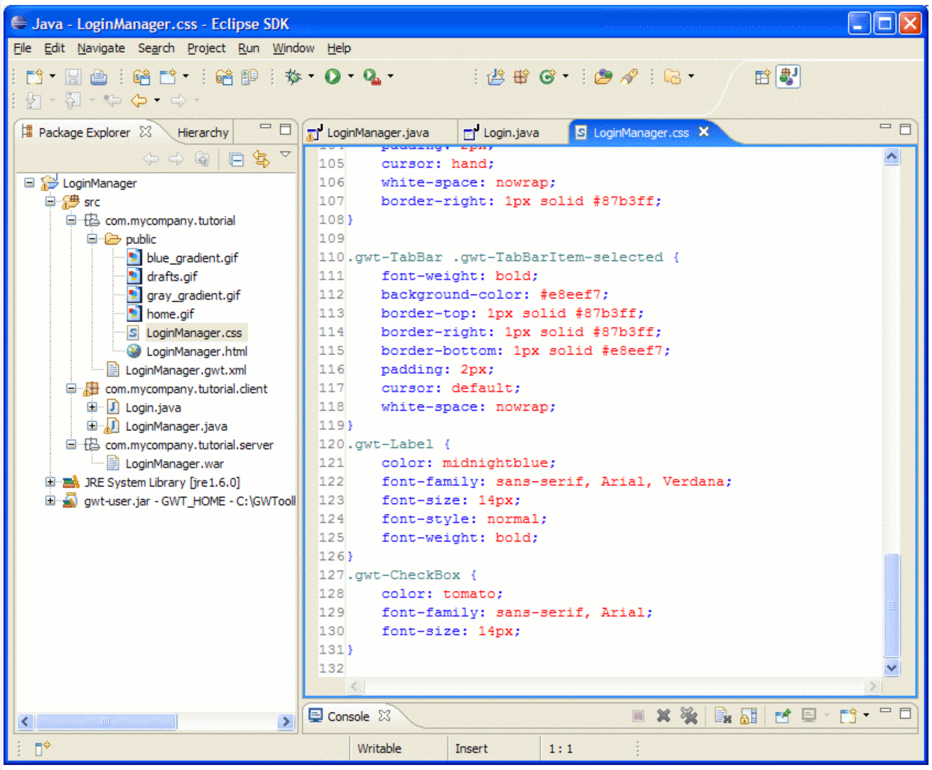


Family: sans-serif, Arial, Verdana
Size: 14 pixels
Style: normal
Variant:
Weight: bold
Stretch:
Color: midnightblue

Decoration: [ ] none [ ] underline [ ] overline [ ] line-through [ ] blink

Nolite mittere margaeritas ante porcas! *Latin proverb*

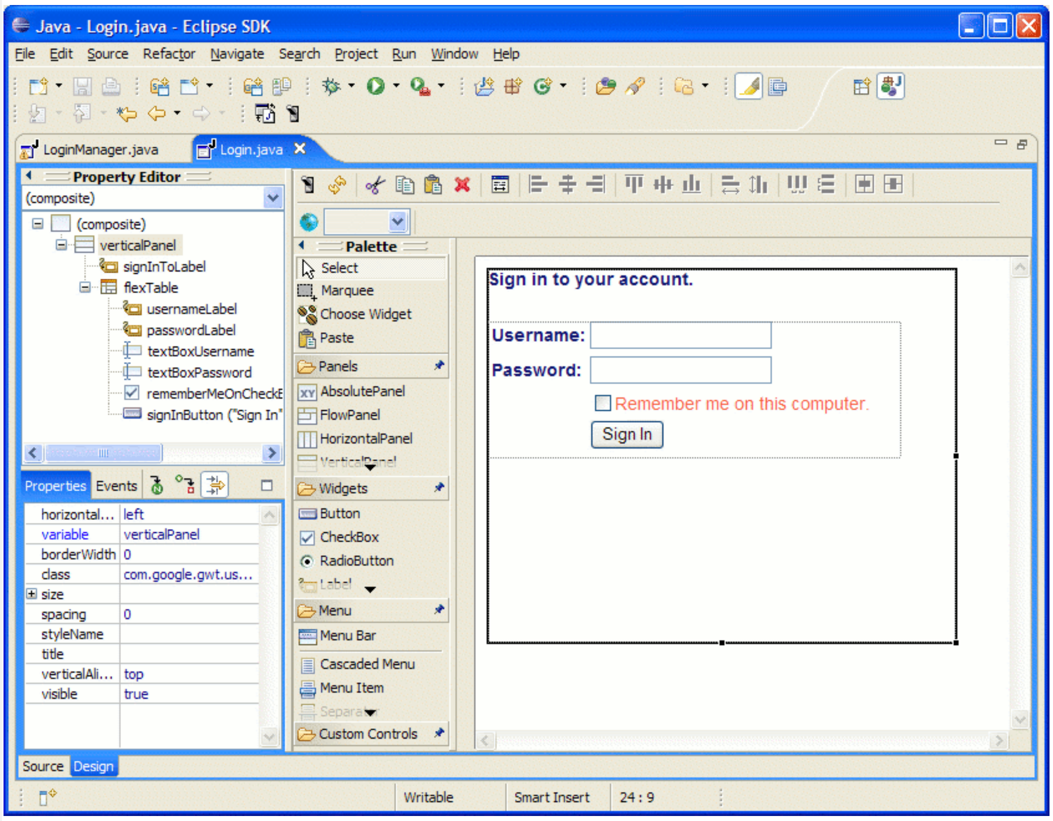
Double-click on the **LoginManager.css** file in your public folder to see the generated css styles.



```
    cursor: hand;
    white-space: nowrap;
    border-right: 1px solid #87b3ff;
}

.gwt-TabBar .gwt-TabBarItem-selected {
    font-weight: bold;
    background-color: #e8eef7;
    border-top: 1px solid #87b3ff;
    border-right: 1px solid #87b3ff;
    border-bottom: 1px solid #e8eef7;
    padding: 2px;
    cursor: default;
    white-space: nowrap;
}
.gwt-Label {
    color: midnightblue;
    font-family: sans-serif, Arial, Verdana;
    font-size: 14px;
    font-style: normal;
    font-weight: bold;
}
.gwt-CheckBox {
    color: tomato;
    font-family: sans-serif, Arial;
    font-size: 14px;
}
```

This is how the Login looks like with some css styling.

Make sure to **Save** your work.

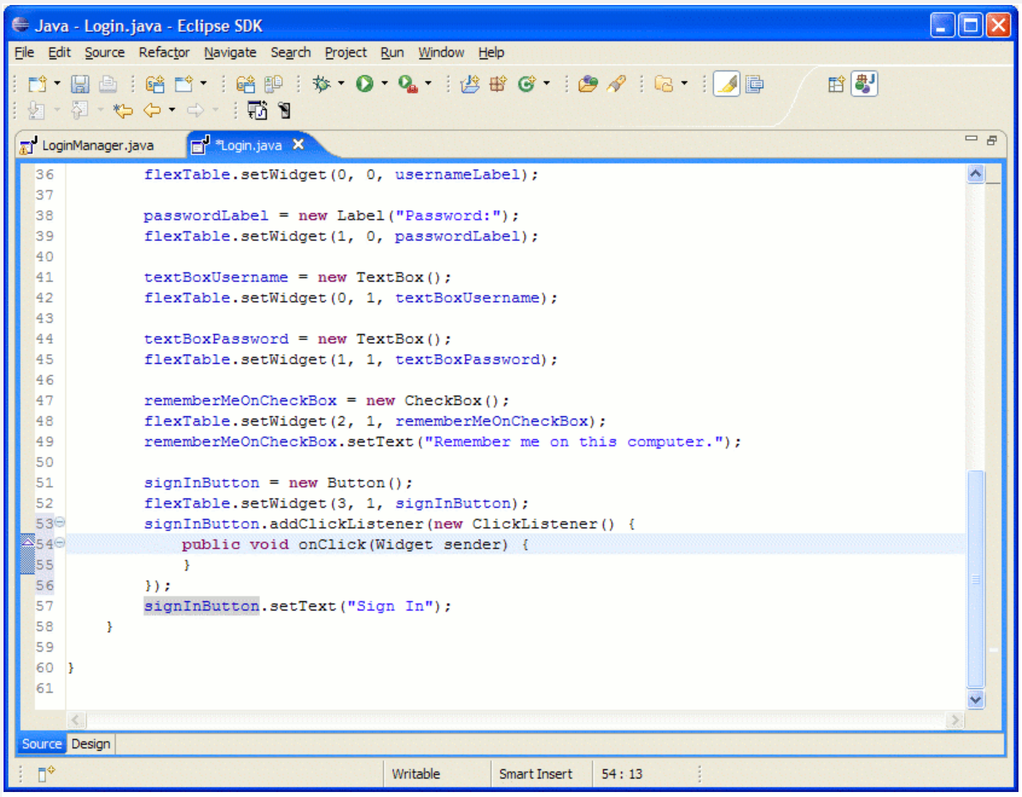


horizontal…	left
variable	verticalPanel
borderWidth	0
class	com.google.gwt.us…
size	
spacing	0
styleName	
title	
verticalAli…	top
visible	true

Add Event handlers.

- Right-click the Sign In button.
- Select **Add event handler** > **click** > **onClick**.

The Designer switched to Source view. As you can see a ClickListener has been added. All you have to do is implement the **onClick** event.

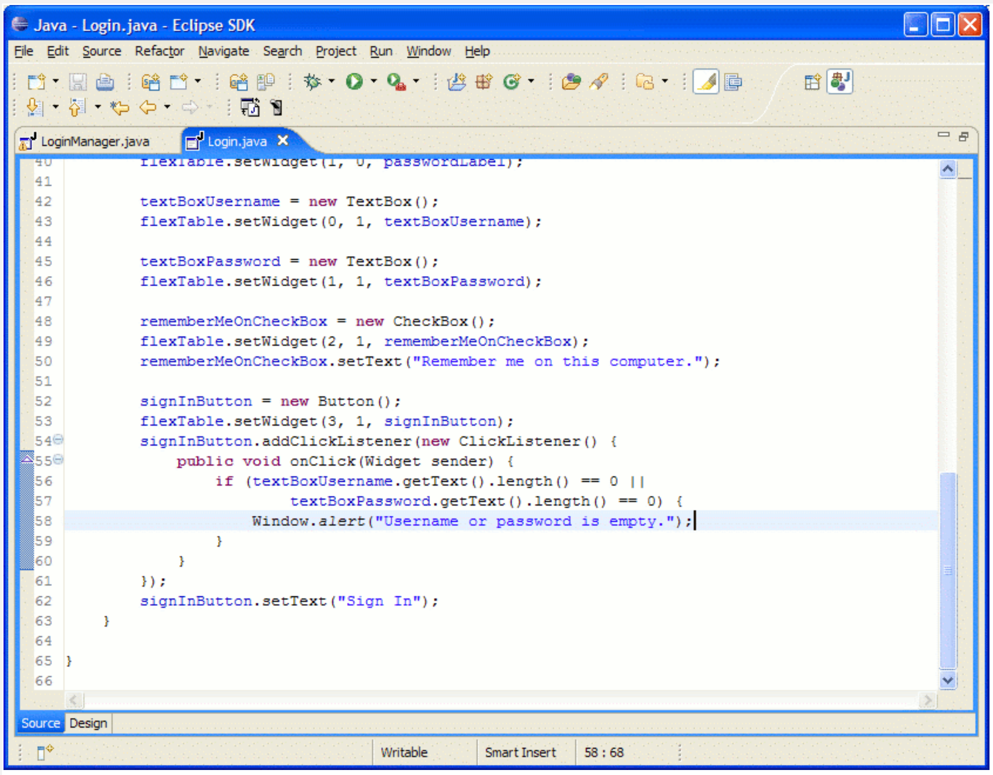

Java - Login.java - Eclipse SDK
File Edit Source Refactor Navigate Search Project Run Window Help
LoginManager.java
*Login.java
Source Design
Writable
Smart Insert
54 : 13

```
        flexTable.setWidget(0, 0, usernameLabel);

        passwordLabel = new Label("Password:");
        flexTable.setWidget(1, 0, passwordLabel);

        textBoxUsername = new TextBox();
        flexTable.setWidget(0, 1, textBoxUsername);

        textBoxPassword = new TextBox();
        flexTable.setWidget(1, 1, textBoxPassword);

        rememberMeOnCheckBox = new CheckBox();
        flexTable.setWidget(2, 1, rememberMeOnCheckBox);
        rememberMeOnCheckBox.setText("Remember me on this computer.");

        signInButton = new Button();
        flexTable.setWidget(3, 1, signInButton);
        signInButton.addClickListener(new ClickListener() {
            public void onClick(Widget sender) {
            }
        });
        signInButton.setText("Sign In");
    }

}
```

Implement the onClick event. Type or copy and paste the following code:

You may need to add an import statement:

```
import com.google.gwt.user.client.Window;

if (textBoxUsername.getText().length() == 0 || textBoxPassword.getText().length() == 0) {
   Window.alert("Username or password is empty.");

}
```

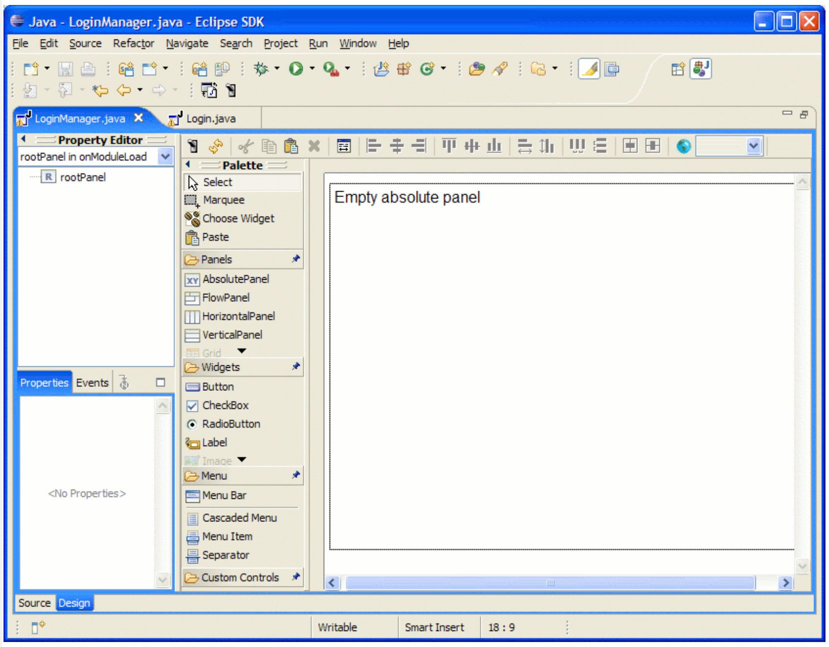

Java - Login.java - Eclipse SDK
File Edit Source Refactor Navigate Search Project Run Window Help
LoginManager.java
Login.java
Source
Design
Writable
Smart Insert
58 : 68

```
        flexTable.setWidget(1, 0, passwordLabel);

        textBoxUsername = new TextBox();
        flexTable.setWidget(0, 1, textBoxUsername);

        textBoxPassword = new TextBox();
        flexTable.setWidget(1, 1, textBoxPassword);

        rememberMeOnCheckBox = new CheckBox();
        flexTable.setWidget(2, 1, rememberMeOnCheckBox);
        rememberMeOnCheckBox.setText("Remember me on this computer.");

        signInButton = new Button();
        flexTable.setWidget(3, 1, signInButton);
        signInButton.addClickListener(new ClickListener() {
            public void onClick(Widget sender) {
                if (textBoxUsername.getText().length() == 0 ||
                        textBoxPassword.getText().length() == 0) {
                    Window.alert("Username or password is empty.");
                }
            }
        });
        signInButton.setText("Sign In");
    }

}
```

5. Add the Login Composite to the LoginManager module.

Open the LoginManager class in **Design mode** and delete the initial Click me! button.

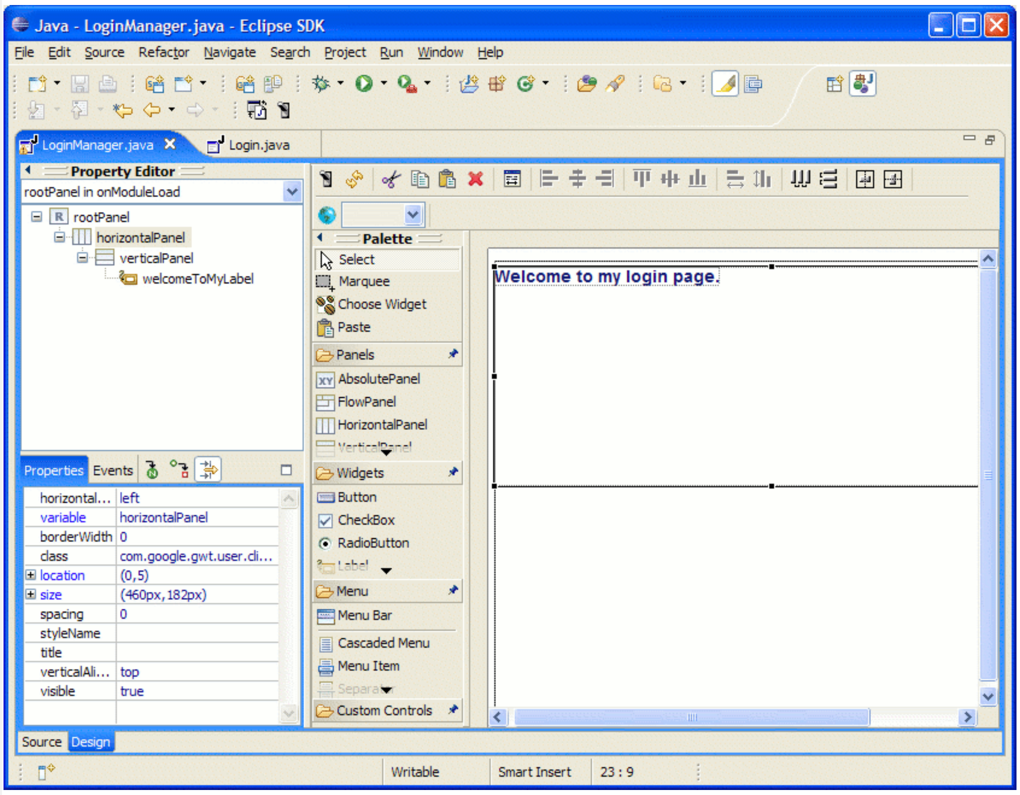
You should now see an empty module with the default rootPanel.



Add a **HorizontalPanel** within the **rootPanel**. Drag the bottom right corner of the HorizontalPanel to make it bigger.

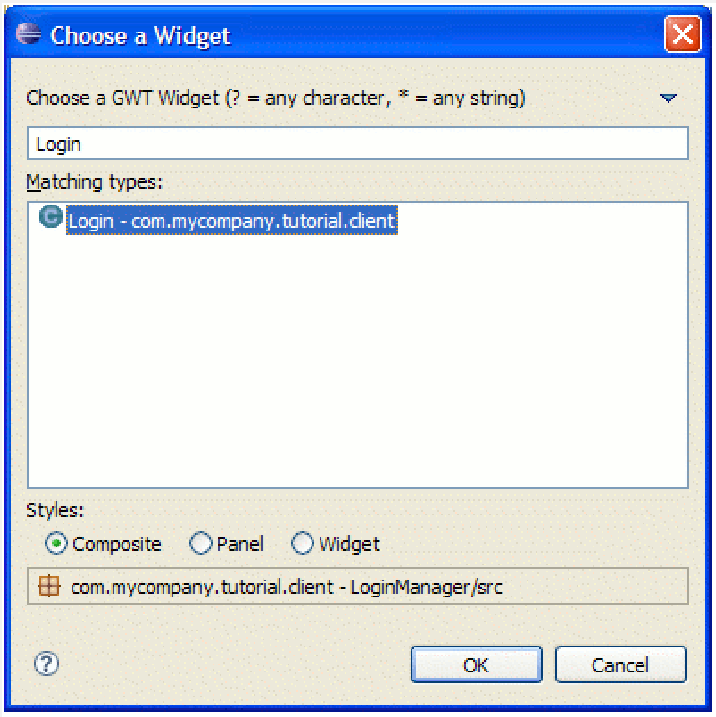
Add a **VerticalPanel** to the **HorizontalPanel**.

Add a Label to the VerticalPanel. Set the text property to **Welcome to my login page**.



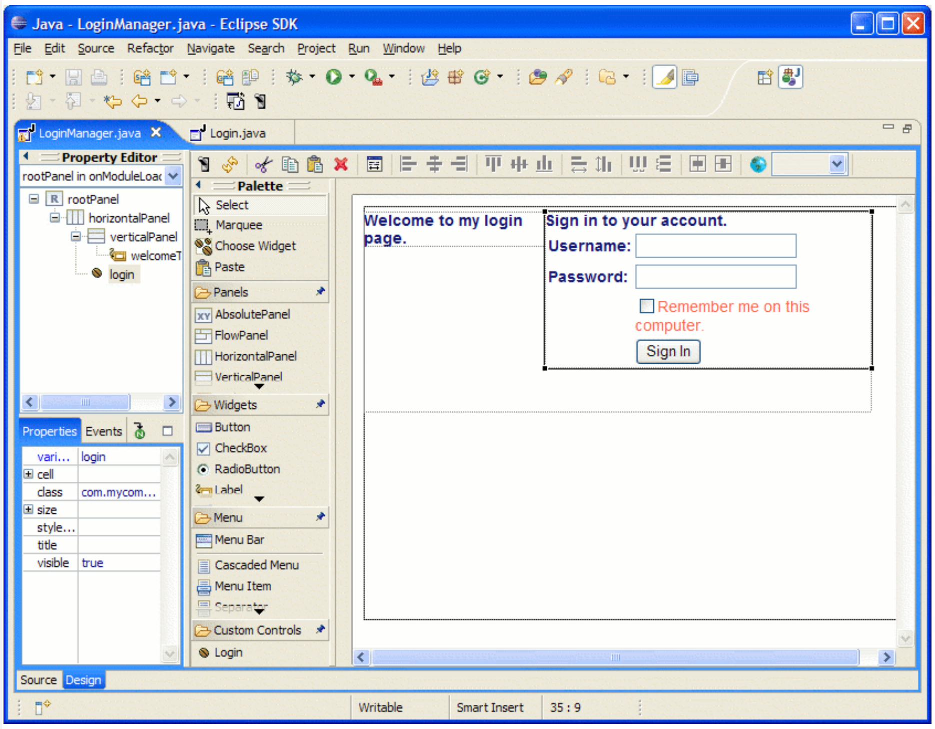
horizontal...	left
variable	horizontalPanel
borderWidth	0
class	com.google.gwt.user.cli...
location	(0,5)
size	(460px, 182px)
spacing	0
styleName	
title	
verticalAli...	top
visible	true

Select **Choose Widget from the Palette.** In the Choose Widget dialog, type **Login** and click OK.


Choose a Widget
Choose a GWT Widget (? = any character, * = any string)
Login
Matching types:
Login - com.mycompany.tutorial.client
Styles:
Composite
Panel
Widget
com.mycompany.tutorial.client - LoginManager/src
OK
Cancel

Place the widget in the LoginManager inside the HorizontalPanel as shown.

Click **Save**.

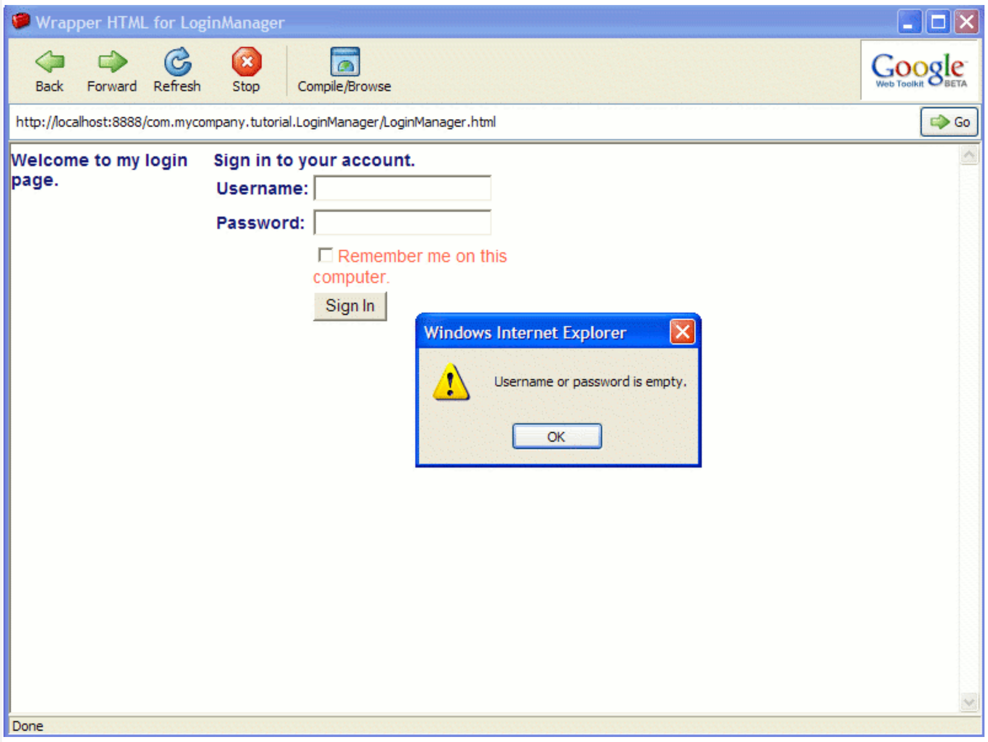


6. Run your Application in Hosted mode.

In the Package explorer, **right-click the LoginManager** and select **Run As > GWT Application**.

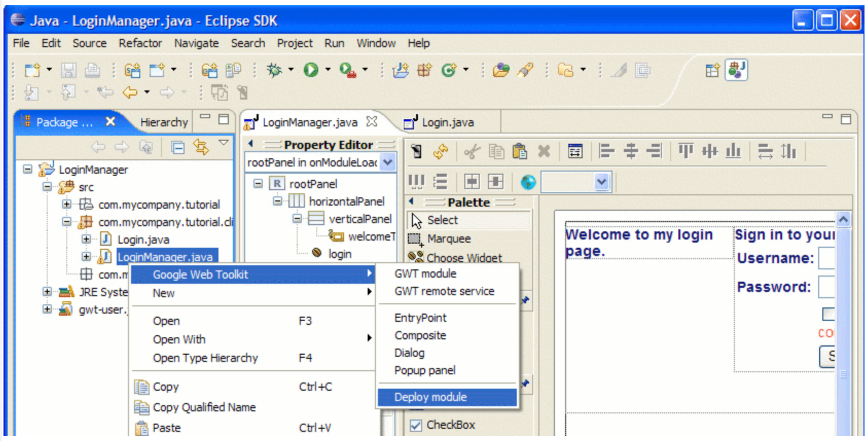
This will create a launch configuration and run the application locally.

Click on the **Sign In button**.


Wrapper HTML for LoginManager
Back
Forward
Refresh
Stop
Compile/Browse
http://localhost:8888/com.mycompany.tutorial.LoginManager/LoginManager.html
Go
Welcome to my login page.
Sign in to your account.
Username:
Password:
Remember me on this computer.
Sign In
Windows Internet Explorer
Username or password is empty.
OK
Done

7. Build and Deploy.

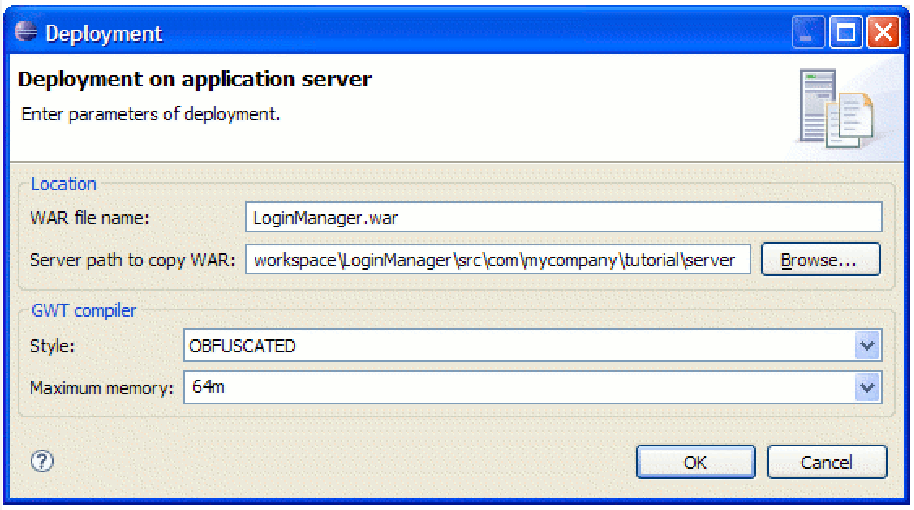
Right click on **LoginManager** and select **Google Web Toolkit > Deploy module**



Specify the server deployment options and click OK.

For more information on deployment, see the GWT Module Deployment documentation.

For more information on deploying to your server, see the Google Website and forums.



Deployment on application server

Enter parameters of deployment.

Location

WAR file name: LoginManager.war

Server path to copy WAR: workspace\LoginManager\src\com\mycompany\tutorial\server

GWT compiler

Style: OBFUSCATED

Maximum memory: 64m

You can download the source code here.

6. Google AppEngine

Getting Started: Java

This tutorial describes how to develop and deploy a simple Java project with Google App Engine. The example project, a guest book, demonstrates how to use the Java runtime environment, and how to use several App Engine services, including the datastore and Google Accounts.

This tutorial has the following sections:

- Introduction
- Installing the Java SDK
- Creating a Project
- Using the Users Service
- Using JSPs
- Using the Datastore with JDO
- Using Static Files
- Uploading Your Application

Introduction

Welcome to Google App Engine! Creating an App Engine application is easy, and only takes a few minutes. And it's free to start: upload your app and share it with users right away, at no charge and with no commitment required.

Google App Engine applications can be written in either the Java or Python programming languages. This tutorial covers **Java**. If you're more likely to use Python to build your applications, see Getting Started: Python.

In this tutorial, you will learn how to:

- build an App Engine application using standard Java web technologies, such as servlets and JSPs
- create an App Engine Java project with Eclipse, and without
- use the Google Plugin for Eclipse for App Engine development
- use the App Engine datastore with the Java Data Objects (JDO) standard interface
- integrate an App Engine application with Google Accounts for user authentication
- upload your app to App Engine

By the end of the tutorial, you will have implemented a working application, a simple guest book that lets users post messages to a public message board.

Installing the Java SDK

You develop and upload Java applications for Google App Engine using the App Engine Java software development kit (SDK).

The SDK includes software for a web server that you can run on your own computer to test your Java applications. The server simulates all of the App Engine services, including a local version of the datastore, Google Accounts, and the ability to fetch URLs and send email from your computer using the App Engine APIs.

Getting Java

Google App Engine supports Java 5 and Java 6. When your Java application is running on App Engine, it runs using the Java 6 virtual machine (JVM) and standard libraries. Ideally, you should use Java 6 for compiling and testing your application to ensure that the local server behaves similarly to App Engine.

For developers that don't have easy access to Java 6 (such as developers using Mac OS X), the App Engine SDK is compatible with Java 5. You can upload compiled classes and JARs made with Java 5 to App Engine.

If necessary, download and install the Java SE Development Kit (JDK) for your platform. Mac users, see Apple's Java developer site to download and install the latest version of the Java Developer Kit available for Mac OS X.

Once the JDK is installed, run the following commands from a command prompt (for Windows, Command Prompt; for Mac OS X, Terminal) to verify that you can run the commands, and to determine which version is installed. If you have Java 6 installed, these commands will report a version number similar to 1.6.0. If you have Java 5 installed, the version number will be similar to 1.5.0.

```
java -version

javac -version
```

Using Eclipse and the Google Plugin for Eclipse

If you are using the Eclipse development environment, the easiest way to develop, test and upload App Engine apps is to use the Google Plugin for Eclipse. The plugin includes everything you need to build, test and deploy your app, entirely within Eclipse.

The plugin is available for Eclipse versions 3.3, 3.4, and 3.5. You can install the plugin using the Software Update feature of Eclipse. The installation locations are as follows:

- The Google Plugin for Eclipse, for Eclipse 3.3 (Europa):

  http://dl.google.com/eclipse/plugin/3.3

- The Google Plugin for Eclipse, for Eclipse 3.4 (Ganymede):

http://dl.google.com/eclipse/plugin/3.4

- The Google Plugin for Eclipse, for Eclipse 3.5 (Galileo):

  http://dl.google.com/eclipse/plugin/3.5

For details on how to use Software Update to install the plugin, and how to create a new project, see Using the Google Eclipse Plugin.

Getting the SDK

If you are using Eclipse and the Google Plugin, you can install the App Engine SDK from Eclipse using Software Update. If you haven't already, install the "Google App Engine Java SDK" component using the locations above.

If you are not using Eclipse or the Google Plugin, you can download the App Engine Java SDK as a Zip archive.

Download the App Engine Java SDK. Unpack the archive in a convenient location on your hard drive.

Note: Unpacking the archive creates a directory whose name is something like appengine-java-sdk-X.X.X, where X.X.X is the SDK version number. Throughout this documentation, this directory will be referred to as appengine-java-sdk/. You may want to rename the directory after unpacking.

Trying a Demo Application

The App Engine Java SDK includes several demo applications in the demos/ directory. The final version of the guest book application you will create in this tutorial is included under the directory guestbook/. This demo has been precompiled for you so you can try it right away.

If you are using Eclipse, the SDK is located in your Eclipse installation directory, under plugins/com.google.appengine.eclipse.sdkbundle_*VERSION*/, where *VERSION* is a version identifier for the SDK. From the command line, change the current working directory to this directory to run the following command. If you're using Mac OS X or Linux, you may need to give the command files executable permissions before you can run them (such as with the command `chmod u+x dev_appserver.sh`).

If you are using Windows, start the guest book demo in the development server by running the following command at a command prompt:

```
appengine-java-sdk\bin\dev_appserver.cmd appengine-java-sdk\demos\guestbook\war
```

If you are using Mac OS X or Linux, run the following command:

```
./appengine-java-sdk/bin/dev_appserver.sh appengine-java-sdk/demos/guestbook/war
```

The development server starts, and listens for requests on port 8080. Visit the following URL in your browser:

- http://localhost:8080/

Note: When you start the development server from within Eclipse using the Google Plugin for Eclipse (discussed later), the server uses the port 8888 by default: http://localhost:8888/

For more information about running the development web server from the command line, including how to change which port it uses, see the Dev Web Server reference.

To stop the server, make sure the command prompt window is active, then press Control-C.

Creating a Project

App Engine Java applications use the Java Servlet standard for interacting with the web server environment. An application's files, including compiled classes, JARs, static files and configuration files, are arranged in a directory structure using the WAR standard layout for Java web applications. You can use any development process you like to develop web servlets and produce a WAR directory. (WAR archive files are not yet supported by the SDK.)

The Project Directory

For this tutorial, we will use a single directory named `Guestbook/` for all project files. A subdirectory named `src/` contains the Java source code, and a subdirectory named `war/` contains the complete application arranged in the WAR format. Our build process compiles the Java source files and puts the compiled classes in the appropriate location in war/.

The complete project directory looks like this:

```
Guestbook/
  src/
    ...Java source code...
    META-INF/
      ...other configuration...
  war/
    ...JSPs, images, data files...
    WEB-INF/
      ...app configuration...
      lib/
        ...JARs for libraries...
      classes/
        ...compiled classes...
```

If you are using Eclipse, create a new project by clicking the New Web Application Project button in the toolbar: Give the project a "Project name" of Guestbook and a "Package" of guestbook. Uncheck "Use Google Web Toolkit," and ensure "Use Google App Engine" is checked. See Using the Google Plugin for Eclipse for more information. The wizard creates the directory structure, and the files described below.

If you are not using Eclipse, create the directory structure described above. As you read each of the files described in this section, create the files using the given locations and names.

You can also copy the new project template included with the SDK, in the appengine-java-sdk/demos/new_project_template/ directory.

The Servlet Class

App Engine Java applications use the Java Servlet API to interact with the web server. An HTTP servlet is an application class that can process and respond to web requests. This class extends either the javax.servlet.GenericServlet class or the javax.servlet.http.HttpServlet class.

Our guest book project begins with one servlet class, a simple servlet that displays a message.

If you are not using the Eclipse plugin, create the directories for the path src/guestbook/, then create the servlet class file described below.

In the directory src/guestbook/, a file named GuestbookServlet.java has the following contents:

```
package guestbook;

import java.io.IOException;
import javax.servlet.http.*;

public class GuestbookServlet extends HttpServlet {
    public void doGet(HttpServletRequest req, HttpServletResponse resp)
            throws IOException {
        resp.setContentType("text/plain");
        resp.getWriter().println("Hello, world");
    }
}
```

The web.xml File

When the web server receives a request, it determines which servlet class to call using a configuration file known as the "web application deployment descriptor." This file is named web.xml, and resides in the war/WEB-INF/ directory in the WAR. WEB-INF/ and web.xml are part of the servlet specification.

In the directory war/WEB-INF/, a file named web.xml has the following contents:

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE web-app PUBLIC
 "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
 "http://java.sun.com/dtd/web-app_2_3.dtd">

<web-app xmlns="http://java.sun.com/xml/ns/javaee" version="2.5">
    <servlet>
        <servlet-name>guestbook</servlet-name>
        <servlet-class>guestbook.GuestbookServlet</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>guestbook</servlet-name>
        <url-pattern>/guestbook</url-pattern>
    </servlet-mapping>
    <welcome-file-list>
        <welcome-file>index.html</welcome-file>
    </welcome-file-list>
</web-app>
```

This web.xml file declares a servlet named guestbook, and maps it to the URL path /guestbook. It also says that whenever the user fetches a URL path that is not already mapped to a servlet and represents a directory path inside the application's WAR, the server should check for a file named index.html in that directory and serve it if found.

The appengine-web.xml File

App Engine needs one additional configuration file to figure out how to deploy and run the application. This file is named appengine-web.xml, and resides in WEB-INF/ alongside web.xml. It includes the registered ID of your application (Eclipse creates this with an empty ID for you to fill in later), the version number of your application, and lists of files that ought to be treated as static files (such as images and CSS) and resource files (such as JSPs and other application data).

In the directory war/WEB-INF/, a file named appengine-web.xml has the following contents:

```
<?xml version="1.0" encoding="utf-8"?>
<appengine-web-app xmlns="http://appengine.google.com/ns/1.0">
    <application></application>
    <version>1</version>
</appengine-web-app>
```

appengine-web.xml is specific to App Engine, and is not part of the servlet standard. You can find XML schema files describing the format of this file in the SDK, in the appengine-java-sdk/docs/ directory. See Configuring an App for more information about this file.

Running the Project

The App Engine SDK includes a web server application you can use to test your application. The server simulates the App Engine environment and services, including sandbox restrictions, the datastore, and the services.

If you are using Eclipse, you can start the development server within the Eclipse debugger. Make sure the project ("Guestbook") is selected, then in the **Run** menu, select **Debug As** > **Web Application**. See Using the Google Plugin for Eclipse for details on creating the debug configuration.

If you are not using Eclipse, see Using Apache Ant for a build script that can build the project and start the development server. To start the server with this build script, enter the following command: `ant runserver` To stop the server, hit Control-C.

Testing the Application

Start the server, then visit the server's URL in your browser. If you're using Eclipse and the Google Eclipse plugin, the server runs using port 8888 by default:

- http://localhost:8888/guestbook

If you're using the `dev_appserver` command to start the server, the default port is `8080`:

- http://localhost:8080/guestbook

For the rest of this tutorial, we'll assume the server is using port `8888`.

The server calls the servlet, and displays the message in the browser.

Using the Users Service

Google App Engine provides several useful services based on Google infrastructure, accessible by applications using libraries included with the SDK. One such service is the Users service, which lets your application integrate with Google user accounts. With the Users service, your users can use the Google accounts they already have to sign in to your application.

Let's use the Users service to personalize this application's greeting.

Using Users

Edit src/guestbook/GuestbookServlet.java as indicated to resemble the following:

```
package guestbook;

import java.io.IOException;
import javax.servlet.http.*;
import com.google.appengine.api.users.User;
import com.google.appengine.api.users.UserService;
import com.google.appengine.api.users.UserServiceFactory;

public class GuestbookServlet extends HttpServlet {
    public void doGet(HttpServletRequest req, HttpServletResponse resp)
              throws IOException {
        UserService userService = UserServiceFactory.getUserService();
        User user = userService.getCurrentUser();

        if (user != null) {
            resp.setContentType("text/plain");
            resp.getWriter().println("Hello, " + user.getNickname());
        } else {
            resp.sendRedirect(userService.createLoginURL(req.getRequestURI()));
        }
    }
}
```

If you are using Eclipse and your development server is running in the debugger, when you save your changes to this file, Eclipse compiles the new code automatically, then attempts to insert the new code into the already-running server. Changes to classes, JSPs, static files and appengine-web.xml are reflected immediately in the running server without needing to restart. If you change web.xml or other configuration files, you must stop and start the server to see the changes.

If you are using Ant, you must stop the server and rebuild the project to see changes made to source code. Changes to JSPs and static files do not require restarting the server.

Rebuild your project and restart the server, if necessary. Test the application by visiting the servlet URL in your browser:

- http://localhost:8888/guestbook

Instead of displaying the message, the server prompts you for an email address. Enter any email address (such as alfred@example.com, then click "Log In." The app displays a message, this time containing the email address you entered.

The new code for the GuestbookServlet class uses the Users API to check if the user is signed in with a Google Account. If not, the user is redirected to the Google Accounts sign-in screen. userService.createLoginURL(...) returns the URL of the sign-in screen. The sign-in facility knows to redirect the user back to the app by the URL passed to createLoginURL(...), which in this case is the URL of the current page.

The development server knows how to simulate the Google Accounts sign-in facility. When run on your local machine, the redirect goes to the page where you can enter any email address to simulate an account sign-in. When run on App Engine, the redirect goes to the actual Google Accounts screen.

You are now signed in to your test application. If you reload the page, the message will display again.

To allow the user to sign out, provide a link to the sign-out screen, generated by the method createLogoutURL(). Note that a sign-out link will sign the user out of all Google services.

Using JSPs

While we could output the HTML for our user interface directly from the Java servlet code, this would be difficult to maintain as the HTML gets complicated. It's better to use a template system, with the user interface designed and implemented in separate files with placeholders and logic to insert data provided by the application. There are many template systems available for Java, any of which would work with App Engine.

For this tutorial, we'll use JavaServer Pages (JSPs) to implement the user interface for the guest book. JSPs are part of the servlet standard. App Engine compiles JSP files in the application's WAR automatically, and maps them to URL paths.

Hello, JSP!

Our guest book app writes strings to an output stream, but this could also be written as a JSP. Let's begin by porting the latest version of the example to a JSP.

In the directory war/, create a file named guestbook.jsp with the following contents:

```
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<%@ page import="com.google.appengine.api.users.User" %>
<%@ page import="com.google.appengine.api.users.UserService" %>
<%@ page import="com.google.appengine.api.users.UserServiceFactory" %>

<html>
  <body>

<%
    UserService userService = UserServiceFactory.getUserService();
    User user = userService.getCurrentUser();
    if (user != null) {
%>
<p>Hello, <%= user.getNickname() %>! (You can
<a href="<%= userService.createLogoutURL(request.getRequestURI()) %>">sign out</a>.)</p>
<%
    } else {
%>
<p>Hello!
<a href="<%= userService.createLoginURL(request.getRequestURI()) %>">Sign in</a>
to include your name with greetings you post.</p>
<%
    }
%>

  </body>
</html>
```

By default, any file in war/ or a subdirectory (other than WEB-INF/) whose name ends in .jsp is automatically mapped to a URL path. The URL path is the path to the .jsp file, including the filename. This JSP will be mapped automatically to the URL /guestbook.jsp.

For the guest book app, we want this to be the application's home page, displayed when someone accesses the URL /. An easy way to do this is to declare in web.xml that guestbook.jsp is the "welcome" servlet for that path.

Edit war/WEB-INF/web.xml and replace the current <welcome-file> element in the <welcome-file-list>. Be sure to remove index.html from the list, as static files take precedence over JSP and servlets.

```
<welcome-file-list>
    <welcome-file>guestbook.jsp</welcome-file>
</welcome-file-list>
```

Tip: If you are using Eclipse, the editor may open this file in "Design" mode. To edit this file as XML, select the "Source" tab at the bottom of the frame.

Stop then start the development server. Visit the following URL:

- http://localhost:8888/

The app displays the contents of guestbook.jsp, including the user nickname if the user is signed in.

When you load a JSP for the first time, the development server converts the JSP into Java source code, then compiles the Java source into Java bytecode. The Java source and the compiled class are saved to a temporary directory. The development server regenerates and compiles JSPs automatically if the original JSP files change.

When you upload your application to App Engine, the SDK compiles all JSPs to bytecode, and only uploads the bytecode. When your app is running on App Engine, it uses the compiled JSP classes.

The Guestbook Form

Our guest book application will need a web form so the user can post a new greeting, and a way to process that form. The HTML of the form will go into the JSP. The destination of the form will be a new URL, /sign, to be handled by a new servlet class, SignGuestbookServlet.

SignGuestbookServlet will process the form, then redirect the user's browser back to /guestbook.jsp. For now, the new servlet will just write the posted message to the log.

Edit guestbook.jsp, and put the following lines just above the closing </body> tag:

```
  ...

  <form action="/sign" method="post">
    <div><textarea name="content" rows="3" cols="60"></textarea></div>
    <div><input type="submit" value="Post Greeting" /></div>
  </form>

  </body>
</html>
```

Create a new class named SignGuestbookServlet in the package guestbook. (Non-Eclipse users, create the file SignGuestbookServlet.java in the directory src/guestbook/.) Give the source file the following contents:

```
package guestbook;

import java.io.IOException;
import java.util.logging.Logger;
import javax.servlet.http.*;
import com.google.appengine.api.users.User;
import com.google.appengine.api.users.UserService;
import com.google.appengine.api.users.UserServiceFactory;

public class SignGuestbookServlet extends HttpServlet {
    private static final Logger log = Logger.getLogger(SignGuestbookServlet.class.getName());

    public void doPost(HttpServletRequest req, HttpServletResponse resp)
                throws IOException {
        UserService userService = UserServiceFactory.getUserService();
        User user = userService.getCurrentUser();

        String content = req.getParameter("content");
        if (content == null) {
            content = "(No greeting)";
        }
        if (user != null) {
            log.info("Greeting posted by user " + user.getNickname() + ": " + content);
        } else {
            log.info("Greeting posted anonymously: " + content);
        }
        resp.sendRedirect("/guestbook.jsp");
    }
}
```

Edit war/WEB-INF/web.xml and add the following lines to declare the SignGuestbookServlet servlet and map it to the the /sign URL:

```
<web-app xmlns="http://java.sun.com/xml/ns/javaee" version="2.5">
    ...

    <servlet>
        <servlet-name>sign</servlet-name>
        <servlet-class>guestbook.SignGuestbookServlet</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>sign</servlet-name>
        <url-pattern>/sign</url-pattern>
    </servlet-mapping>

    ...
</web-app>
```

This new servlet uses the java.util.logging.Logger class to write messages to the log. You can control the behavior of this class using a logging.properties file, and a system property set in the app's appengine-web.xml file. If you are using Eclipse, your app was created with a default version of this file in your app's src/ and the appropriate system property.

If you are not using Eclipse, you must set up the Logger configuration file manually. Copy the example file from the SDK appengine-java-sdk/config/user/logging.properties to your app's war/WEB-INF/ directory. Then edit the app's war/WEB-INF/appengine-web.xml file as indicated:

```
<appengine-web-app xmlns="http://appengine.google.com/ns/1.0">
    ...

    <system-properties>
```

```
        <property name="java.util.logging.config.file" value="WEB-INF/logging.properties"/>
    </system-properties>

</appengine-web-app>
```

The servlet logs messages using the INFO log level (using log.info()). The default log level is WARNING, which suppresses INFO messages from the output. To change the log level for all classes in the guestbook package, edit the logging.properties file and add an entry for guestbook.level, as follows:

```
.level = WARNING
guestbook.level = INFO

...
```

Tip: When your app logs messages using the java.util.logging.Logger API while running on App Engine, App Engine records the messages and makes them available for browsing in the Admin Console, and available for downloading using the AppCfg tool. The Admin Console lets you browse messages by log level.

Rebuild and restart, then test http://localhost:8888/. The form displays. Enter some text in the form, and submit. The browser sends the form to the app, then redirects back to the empty form. The greeting data you entered is logged to the console by the server.

Using the Datastore with JDO

Storing data in a scalable web application can be tricky. A user could be interacting with any of dozens of web servers at a given time, and the user's next request could go to a different web server than the one that handled the previous request. All web servers need to be interacting with data that is also spread out across dozens of machines, possibly in different locations around the world.

With Google App Engine, you don't have to worry about any of that. App Engine's infrastructure takes care of all of the distribution, replication and load balancing of data behind a simple API—and you get a powerful query engine and transactions as well.

The App Engine datastore is one of several services provided by App Engine with two APIs: a standard API, and a low-level API. By using the standard APIs, you make it easier to port your application to other hosting environments and other database technologies, if you ever need to. Standard APIs "decouple" your application from the App Engine services. App Engine services also provide low-level APIs that exposes the service capabilities directly. You can use the low-level APIs to implement new adapter interfaces, or just use the APIs directly in your app.

App Engine includes support for two different API standards for the datastore: Java Data Objects (JDO) and Java Persistence API (JPA). These interfaces are provided by DataNucleus Access Platform, an open source implementation of several Java persistence standards, with an adapter for the App Engine datastore.

For the guest book, we'll use the JDO interface to retrieve and post messages left by users.

Setting Up DataNucleus Access Platform

Access Platform needs a configuration file that tells it to use the App Engine datastore as the backend for the JDO implementation. In the final WAR, this file is named jdoconfig.xml and resides in the directory war/WEB-INF/classes/META-INF/.

If you are using Eclipse, this file has been created for you as src/META-INF/jdoconfig.xml. This file is automatically copied into war/WEB-INF/classes/META-INF/ when you build your project.

If you are not using Eclipse, you can create the directory war/WEB-INF/classes/META-INF/ directly, or have your build process create it and copy the configuration file from another location. The Ant build script described in Using Apache Ant copies this file from src/META-INF/.

The jdoconfig.xml file should have the following contents:

```
<?xml version="1.0" encoding="utf-8"?>
<jdoconfig xmlns="http://java.sun.com/xml/ns/jdo/jdoconfig"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:noNamespaceSchemaLocation="http://java.sun.com/xml/ns/jdo/jdoconfig">

   <persistence-manager-factory name="transactions-optional">
       <property name="javax.jdo.PersistenceManagerFactoryClass"
           value="org.datanucleus.store.appengine.jdo.DatastoreJDOPersistenceManagerFactory"/>
```

```
        <property name="javax.jdo.option.ConnectionURL" value="appengine"/>
        <property name="javax.jdo.option.NontransactionalRead" value="true"/>
        <property name="javax.jdo.option.NontransactionalWrite" value="true"/>
        <property name="javax.jdo.option.RetainValues" value="true"/>
        <property name="datanucleus.appengine.autoCreateDatastoreTxns" value="true"/>
    </persistence-manager-factory>
</jdoconfig>
```

JDO Class Enhancement

When you create your JDO classes, you use Java annotations to describe how instances should be stored in the datastore, and how they should be recreated when retrieved from the datastore. Access Platform connects your data classes to the implementation using a post-compilation processing step, which DataNucleus calls "enhancing" the classes.

If you are using Eclipse and the Google Plugin, the plugin performs the JDO class enhancement step automatically as part of the build process.

If you are using the Ant build script described in Using Apache Ant, the build script includes the necessary enhancement step.

For more information on JDO class enhancement, see Using JDO.

POJOs and JDO Annotations

JDO allows you to store Java objects (sometimes called Plain Old Java Objects, or POJOs) in any datastore with a JDO-compliant adapter, such as DataNucleus Access Platform. The App Engine SDK includes an Access Platform plugin for the App Engine datastore. This means you can store instances of classes you define in the App Engine datastore, and retrieve them as objects using the JDO API. You tell JDO how to store and reconstruct instances of your class using Java annotations.

Let's create a Greeting class to represent individual messages posted to the guest book.

Create a new class named Greeting in the package guestbook. (Non-Eclipse users, create the file Greeting.java in the directory src/guestbook/.) Give the source file the following contents:

```
package guestbook;

import com.google.appengine.api.datastore.Key;
import com.google.appengine.api.users.User;

import java.util.Date;
import javax.jdo.annotations.IdGeneratorStrategy;
import javax.jdo.annotations.PersistenceCapable;
import javax.jdo.annotations.Persistent;
import javax.jdo.annotations.PrimaryKey;

@PersistenceCapable
public class Greeting {
   @PrimaryKey
   @Persistent(valueStrategy = IdGeneratorStrategy.IDENTITY)
   private Key key;

   @Persistent
   private User author;

   @Persistent
   private String content;

   @Persistent
   private Date date;

   public Greeting(User author, String content, Date date) {
      this.author = author;
      this.content = content;
      this.date = date;
   }

   public Key getKey() {
      return key;
   }

   public User getAuthor() {
      return author;
   }
```

```
    public String getContent() {
        return content;
    }

    public Date getDate() {
        return date;
    }

    public void setAuthor(User author) {
        this.author = author;
    }

    public void setContent(String content) {
        this.content = content;
    }

    public void setDate(Date date) {
        this.date = date;
    }
}
```

This simple class defines 3 properties for a greeting: author, content and date. These three private fields are annotated with @Persistent to tell DataNucleus to store them as properties of objects in the App Engine datastore.

This class defines getters and setters for the properties, which are used only by the application. Using setters is the simplest way of ensuring that the JDO implementation recognizes your updates. Modifying the fields directly bypasses a feature of JDO that saves updated fields automatically, unless you make other changes to your code to enable this

The class also defines a field named key, a Key annotated as both @Persistent and @PrimaryKey. The App Engine datastore has a notion of entity keys, and can represent the key in several different ways on the object. The Key class represents all aspects of App Engine datastore keys, including a numeric ID that is set automatically to a unique value when the object is saved.

For more information on JDO annotations, see Defining Data Classes.

The PersistenceManagerFactory

Each request that uses the datastore creates a new instance of the PersistenceManager class. It does so using an instance of the PersistenceManagerFactory class.

A PersistenceManagerFactory instance takes time to initialize. Thankfully, you only need one instance for your application, and this instance can be stored in a static variable to be used by multiple requests and multiple classes. An easy way to do this is to create a singleton wrapper class for the static instance.

Create a new class named PMF in the package guestbook (a file named PMF.java In the directory src/guestbook/), and give it the following contents:

```
package guestbook;

import javax.jdo.JDOHelper;
import javax.jdo.PersistenceManagerFactory;

public final class PMF {
    private static final PersistenceManagerFactory pmfInstance =
        JDOHelper.getPersistenceManagerFactory("transactions-optional");

    private PMF() {}

    public static PersistenceManagerFactory get() {
        return pmfInstance;
    }
}
```

Creating and Saving Objects

With DataNucleus and the Greeting class in place, the form processing logic can now store new greetings in the datastore.

Edit src/guestbook/SignGuestbookServlet.java as indicated to resemble the following:

```
package guestbook;
```

```
import java.io.IOException;
import java.util.Date;
import java.util.logging.Logger;
import javax.jdo.PersistenceManager;
import javax.servlet.http.*;
import com.google.appengine.api.users.User;
import com.google.appengine.api.users.UserService;
import com.google.appengine.api.users.UserServiceFactory;

import guestbook.Greeting;
import guestbook.PMF;

public class SignGuestbookServlet extends HttpServlet {
    private static final Logger log = Logger.getLogger(SignGuestbookServlet.class.getName());

    public void doPost(HttpServletRequest req, HttpServletResponse resp)
                throws IOException {
        UserService userService = UserServiceFactory.getUserService();
        User user = userService.getCurrentUser();

        String content = req.getParameter("content");
        Date date = new Date();
        Greeting greeting = new Greeting(user, content, date);

        PersistenceManager pm = PMF.get().getPersistenceManager();
        try {
            pm.makePersistent(greeting);
        } finally {
            pm.close();
        }

        resp.sendRedirect("/guestbook.jsp");
    }
}
```

This code creates a new Greeting instance by calling the constructor. To save the instance to the datastore, it creates a PersistenceManager using a PersistenceManagerFactory, then passes the instance to the PersistenceManager's makePersistent() method. The annotations and bytecode enhancement take it from there. Once makePersistent() returns, the new object is stored in the datastore.

Queries With JDOQL

The JDO standard defines a mechanism for querying persistent objects called JDOQL. You can use JDOQL to perform queries of entities in the App Engine datastore, and retrieve results as JDO-enhanced objects.

For this example, we will keep things simple by writing the query code directly into guestbook.jsp. For a larger application, you may want to delegate the query logic to another class.

Edit war/guestbook.jsp and add the indicated lines so that it resembles the following:

```
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<%@ page import="java.util.List" %>
<%@ page import="javax.jdo.PersistenceManager" %>
<%@ page import="com.google.appengine.api.users.User" %>
<%@ page import="com.google.appengine.api.users.UserService" %>
<%@ page import="com.google.appengine.api.users.UserServiceFactory" %>
<%@ page import="guestbook.Greeting" %>
<%@ page import="guestbook.PMF" %>

<html>
  <body>

<%
    UserService userService = UserServiceFactory.getUserService();
    User user = userService.getCurrentUser();
    if (user != null) {
%>
<p>Hello, <%= user.getNickname() %>! (You can
<a href="<%= userService.createLogoutURL(request.getRequestURI()) %>">sign out</a>.)</p>
<%
    } else {
%>
<p>Hello!
<a href="<%= userService.createLoginURL(request.getRequestURI()) %>">Sign in</a>
to include your name with greetings you post.</p>
<%
    }
```

```
%>

<%
    PersistenceManager pm = PMF.get().getPersistenceManager();
    String query = "select from " + Greeting.class.getName();
    List<Greeting> greetings = (List<Greeting>) pm.newQuery(query).execute();
    if (greetings.isEmpty()) {
%>
<p>The guestbook has no messages.</p>
<%
    } else {
        for (Greeting g : greetings) {
            if (g.getAuthor() == null) {
%>
<p>An anonymous person wrote:</p>
<%
            } else {
%>
<p><b><%= g.getAuthor().getNickname() %></b> wrote:</p>
<%
            }
%>
<blockquote><%= g.getContent() %></blockquote>
<%
        }
    }
    pm.close();
%>

    <form action="/sign" method="post">
      <div><textarea name="content" rows="3" cols="60"></textarea></div>
      <div><input type="submit" value="Post Greeting" /></div>
    </form>

  </body>
</html>
```

To prepare a query, you call the newQuery() method of a PersistenceManager instance with the text of the query as a string. The method returns a query object. The query object's execute()

method performs the query, then returns a List<> of result objects of the appropriate type. The query string must include the full name of the class to query, including the package name.

Rebuild the project and restart the server. Visit http://localhost:8888/. Enter a greeting and submit it. The greeting appears above the form. Enter another greeting and submit it. Both greetings are displayed. Try signing out and signing in using the links, and try submitting messages while signed in and while not signed in.

Tip: In a real world application, it's a good idea to escape HTML characters when displaying user-submitted content, like the greetings in this app. The JavaServer Pages Standard Tag Library (JSTL) includes routines for doing this. App Engine includes the JSTL (and other JSP-related runtime JARs), so you do not need to include them with your app. Look for the escapeXml function in the tag library http://java.sun.com/jsp/jstl/functions. See Sun's J2EE 1.4 Tutorial for more information.

Introducing JDOQL

Our guestbook currently displays all messages ever posted to the system. It also displays them in the order they were created. When our guestbook has many messages, it might be more useful to display only recent messages, and display the most recent message at the top. We can do this by adjusting the datastore query.

You perform a query with the JDO interface using JDOQL, a SQL-like query language for retrieving data objects. In our JSP page, the following line defines the JDOQL query string:

```
    String query = "select from " + Greeting.class.getName();
```

In other words, the JDOQL query string is the following:

```
select from guestbook.Greeting
```

This query asks the datastore for every instance of the Greeting class saved so far.

A query can specify the order in which the results ought to be returned in terms of property values. To retrieve all Greeting objects in the reverse of the order in which they were posted (newest to oldest), you would use the following query:

```
select from guestbook.Greeting order by date desc
```

A query can limit the results returned to a range of results. To retrieve just the 5 most recent greetings, you would use `order by` and `range` together, as follows:

```
select from guestbook.Greeting order by date desc range 0,5
```

Let's do this for the guest book application. In `guestbook.jsp`, replace the `query` definition with the following:

```
    String query = "select from " + Greeting.class.getName() + " order by date desc range 0,5";
```

Post greetings to the guest book until there are more than 5. Only the most recent 5 are displayed, in reverse chronological order.

For more information about queries and JDOQL, see Queries and Indexes.

Using Static Files

There are many cases where you want to serve static files directly to the web browser. Images, CSS stylesheets, JavaScript code, movies and Flash animations are all typically served directly to the browser. For efficiency, App Engine serves static files from separate servers than those that invoke servlets.

By default, App Engine makes all files in the WAR available as static files except JSPs and files in `WEB-INF/`. Any request for a URL whose path matches a static file serves the file directly to the browser—even if the path also matches a servlet or filter mapping. You can configure which files App Engine treats as static files using the `appengine-web.xml` file.

Let's spruce up our guest book's appearance with a CSS stylesheet. For this example, we will not change the configuration for static files. See App Configuration for more information on configuring static files and resource files.

A Simple Stylesheet

In the directory war/, create a directory named stylesheets/. In this directory, create a file named main.css with the following contents:

```
body {
    font-family: Verdana, Helvetica, sans-serif;
    background-color: #FFFFCC;
}
```

Edit war/guestbook.jsp and insert the following lines just after the <html> line at the top:

```
<html>
  <head>
    <link type="text/css" rel="stylesheet" href="/stylesheets/main.css" />
  </head>

  <body>
    ...
  </body>
</html>
```

Visit http://localhost:8888/. The new version uses the stylesheet.

Uploading Your Application

You create and manage applications in App Engine using the Administration Console. Once you have registered an application ID for your application, you upload it to App Engine using either the Eclipse plugin, or a command-line tool in the SDK.

Note: Once you register an application ID, you can delete it, but you can't re-register that same application ID after it has been deleted. You can skip these next steps if you don't want to register an ID at this time.

Registering the Application

You create and manage App Engine web applications from the App Engine Administration Console, at the following URL:

https://appengine.google.com/

Sign in to App Engine using your Google account. If you do not have a Google account, you can create a Google account with an email address and password.

To create a new application, click the "Create an Application" button. Follow the instructions to register an application ID, a name unique to this application. If you elect to use the free appspot.com domain name, the full URL for the application will be http://*application-id*.appspot.com/. You can also purchase a top-level domain name for your app, or use one that you have already registered.

Edit the appengine-web.xml file, then change the value of the <application> element to be your registered application ID.

Uploading the Application

You can upload your application using Eclipse, or using a command at the command prompt.

Uploading From Eclipse

You can upload your application code and files from within Eclipse using the Google Plugin.

To upload your application from Eclipse, click the App Engine deploy button on the toolbar:

Enter your Google account username (your email address) and password when prompted, then click the **Upload** button. Eclipse gets the application ID and version information from the appengine-web.xml file, and uploads the contents of the war/ directory.

Uploading Using the Command Prompt

You can upload your application code and files using a command included in the SDK named appcfg.cmd (Windows) or appcfg.sh (Mac OS X, Linux).

AppCfg is a multi-purpose tool for interacting with your app on App Engine. The command takes the name of an action, the path to your app's war/ directory, and other options. To upload the app code and files to App Engine, you use the update action.

To upload the app, using Windows:

```
..\appengine-java-sdk\bin\appcfg.cmd update war
```

To upload the app, using Mac OS X or Linux:

```
../appengine-java-sdk/bin/appcfg.sh update war
```

Enter your Google username and password at the prompts.

Accessing Your Application

You can now see your application running on App Engine. If you set up a free appspot.com domain name, the URL for your website begins with your application ID:

http://*application-id*.appspot.com/

Bibliographie

http://code.google.com/intl/fr/appengine/docs/java/gettingstarted/

http://download.instantiations.com/DesignerDoc/integration/latest/docs/html/gwt/tutorial/tutorial_login.html

http://bmoussaud.developpez.com/tutoriel/java/jdo/
http://www.datanucleus.org/products/accessplatform_1_1/guides/eclipse/index.html
http://code.google.com/intl/fr/webtoolkit/tools/gwtdesigner/tutorials/loginmanager.html
http://www.vogella.de/articles/GWT/article.html
http://hbase.apache.org/docs/r0.20.6/cygwin.html
http://hbase.apache.org/docs/r0.20.6/api/overview-summary.html#overview_description
http://code.google.com/intl/fr/appengine/docs/java/gettingstarted/