

Version

1

CANADA'S MICHAEL SMITH

Genome Sciences Centre, BCCA

Chinook Developer Guide

GENOME SCIENCES CENTRE, BCCA

Chinook Developer Guide

© Genome Sciences Centre
BC Cancer Agency
Suite 100
570 West 7th Ave
Vancouver, BC
V5Z 4S6
Phone: (604) 707-5800
First Floor Fax: (604) 876-3561
Fifth Floor Fax: (604) 733-9481

Table of Contents

1.0 Introduction	1
1.0.1 Architecture Overview	1
1.0.2 Use Cases (When/who to use Chinook)	2
2.0 Project Structure	4
2.0.1 Code	4
2.0.2 Libraries	4
2.0.3 Resources	4
2.0.4 Ant.....	5
2.0.5 Classes	5
2.0.6 Log4j	6
2.0.7 Junit	6
3.0 Setting Up	7
3.0.1 Building an Eclipse environment	7
3.0.1.1 Creating a new project using local files	7
3.0.1.2 Creating a project from CVS.....	11
3.0.1.3 Setup environment.....	15
3.0.2 Building a JBuilder environment.....	16
3.0.3 Running a p2p node from an IDE.....	16
3.0.4 Running a server from an IDE.....	17
3.0.5 Running a client from an IDE	19
3.0.6 Running the Perl code from an IDE.....	19
4.0 Detailed Architecture	20
4.0.1 Package Structure.....	20
4.0.1.1. Client packages	20
4.0.1.2. Server packages.....	20
4.0.1.3. p2p packages	21
4.0.1.4. batching packages.....	21
4.0.1.5. parsing packages.....	21
4.0.1.6. common packages.....	22
4.0.2 Managers	22
4.0.2.1 Batch Manager	22

4.0.2.2 Client Manager	22
4.0.2.3 Job Manager	22
4.0.2.4 Service Manager.....	22
4.0.2.5 Discovery Manager.....	22
4.0.2.6 P2P Manager.....	23
4.0.2.2 Server Manager.....	23
4.0.3 Network (sockets)	23
4.0.4 XML.....	23
4.0.5 Database.....	23
4.0.6 GUI.....	23
4.0.7 RMI/Web Services	23
4.0.8 Perl Batching.....	24
4.0.9 Filewire.....	24
4.0.10 Logging	24
4.0.11 Junit	24
5.0 Walkthrough	25
5.0.1 Extending the GUI.....	25
5.0.2 Adding new PERL/JAVA commands.....	25
5.0.3 Adding a new Database	25
5.0.4 Adding a new Data Entry Object Type	25
5.0.5 Deploying Chinook over Webstart.....	25
5.0.6 Adding a new Junit test	27
6.0 Further Information.....	31
6.0.1 Mailing List	31
6.0.2 Authors.....	31
6.0.3 Known Problems	32
6.0.4 License.....	32

1.0 Introduction

Chinook is a peer-to-peer (P2P) bioinformatics platform. The goal of the Chinook platform is to facilitate exchange of analysis techniques within a local community and/or worldwide. Chinook operates by turning command-line applications into services which are broadcast over a virtual network. Currently, there are multiple analysis services that have been made accessible by Chinook. These range from alignment to regulation prediction algorithms. Furthermore, Chinook is designed to make it extremely easy to add new services. This is facilitated using XML (A GUI is under development to facilitate service configuration when this manual is written).

Chinook clients can be operated from Java, Perl, or within applications like [Sockeye](#) (And soon [Pegasys](#) at the Ouellette Lab in Vancouver, and OrthoSeq project at the [Wasserman Lab](#) in Vancouver (CMMI)).

This document is a guide to developers who want to go beyond merely using the Chinook platform, and want to extend and improve Chinook itself.

1.0.1 Architecture Overview

The Chinook network is established using the [JXTA](#) protocol. Each user or service provider on startup attempts to register to the Chinook peer group. The discovery of Chinook peers across the Internet is facilitated by a special type of peer called a rendezvous peer; rendezvous peers are responsible for keeping a list of previously discovered peers. Finding other members on the Chinook network can be facilitated by static rendezvous peers or through dynamic discovery mechanisms. By using these discovery mechanisms, service providers are able to launch Chinook servers within the confines of a few computers, a large-scale grid network, or the Internet.

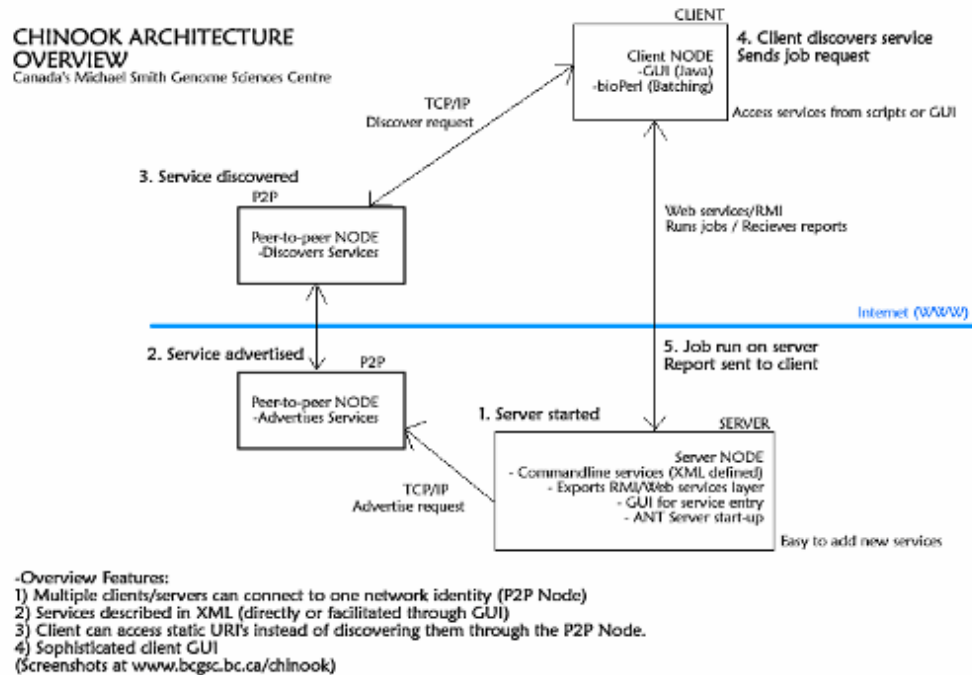


FIGURE 1.1 Chinook Architecture Overview

1.0.2 Use Cases (When/who to use Chinook)

Bioinformatics techniques are used to identify complex, re-occurring relationships in genetic data. Genome sequencing projects and high-throughput expression analyses have contributed large amounts of data; both complicating analysis and demanding higher-level coordination of computational resources. Furthermore, the variety of available bioinformatics tools and algorithms, and their diverse modes of usage create a situation where most users have trouble discerning where to invest their time and resources. Chinook resolves these issues by creating a virtual network for bioinformatics analyses. A user is able to dynamically resolve available bioinformatics services (algorithms) over the Internet or their local network. The user can then validate a server's authenticity and submit bioinformatics analyses to peers that publish their ability to perform desired services. Information like bandwidth, jobs in queue and the location of Chinook services are reported to clients to aid in their job submission process. A user is also able to visit the service creator's website to identify what the particular service does. Chinook allows a service provider to create a new service by simply editing an XML file; as long as the new service has a standard output format, no additional programming is required. The Chinook server runs over the JXTA peer-to-peer network in both Java Remote Method Invocation (RMI) mode or through Apache Axis web services. Chinook creates a virtual community where researchers can rapidly hone in on applications of interest to run them across multiple service providers while shifting the responsibility for application maintenance from the client to the developer. The Chinook virtual network will be used to facilitate high-throughput gene

regulatory analyses and the dynamic multiple alignment of co-expressed and orthologous genes.

Chinook is currently designed to work with only command-line applications where sequence is the input. Chinook groups these applications together to provide a GUI for users to easily run these applications. Current input sequence formats supported are Fasta and Multi-fasta. Work is in progress to allow Chinook to access files uploaded from clients. Furthermore, Chinook has been designed to allow servers to 'plug-in' multiple databases, giving clients access to protein, DNA, or other types of biological data.

2.0 Project Structure

The Chinook project is compartmentalized into client and server side packages and a common package that address peer-to-per functionality. Batch functionality is added to the Chinook platform, so the client can run batch jobs using batch Perl. There are several external directories as well. In this section, we will describe the Chinook packages, the external directories and some of the files that they contain.

2.0.1 Code

The Chinook source code is under **src/** directory. Chinook project package names are created by convention. All the packages are under **ca.bcgsc.chinook** package. The source code addressing client functionality is under **ca.bcgsc.chinook.client** package. The source code addressing server functionality is under **ca.bcgsc.chinook.server** package. The source code under **ca.bcgsc.chinook.p2p** package is used to address peer-to-peer functionality. The source code under **ca.bcgsc.chinook.batching** is used to address batch functionality of client. For more detailed description, go to [4.0.1 Package Structure](#).

2.0.2 Libraries

The Chinook libraries are under **lib/** directory. The lib directory is one of the most important external directories. All the dependency jars in this directory need to be added to the CLASSPATH before running Chinook. If you want to add an external jar to the Chinook project, this directory is where you suppose to put the jar in.

2.0.3 Resources

All the xml files used to configure the Chinook platform are under the **resources** directory. When Chinook starts, it uses the xml files under the resources directory to configure itself. You can customize Chinook platform by modifying files under this directory. Some of the xml files under this directory are very important.

advertisement-config.xml

applications.xml

batch-config.xml

chinookRMI.policy

client-config.xml

database-config.xml

data-entry-config.xml

data-entry-type-config.xml

dbpool.properties

deploy.wsdd

deploy-filewire.wsdd

filewire.xml

filewire-version.xml

log4j.chinook.properties

log4j.filewire.properties

server-config.xml

server-data-entry-config.xml

server-info.xml

static-services.xml

undeploy.wsdd

undeploy-filewire.wsdd

2.0.4 Ant

[Apache Ant](#) is a Java-based build tool. The configuration file is an xml file, which gives Ant the ability to run platform independent. If you haven't used ant before, you may find the [Ant manual](#) is useful. Ant is used as the build tool for Chinook project. The **build.xml** is under the root of the project.

2.0.5 Classes

All the classes compiled from the source code are put in the corresponding packages in the **classes/** directory. By default, when you checkout code from CVS, the classes directory is not there. You need to setup the environment according to the IDE you are using. For more information, refer to [3.0.1 Building an Eclipse Environment](#) or [3.0.2 Building a JBuilder Environment](#).

2.0.6 Log4j

Technically speaking, inserting log statements into the source code will increase its size and reduce its speed. To solve this problem, we are using log4j to insert log statements into Chinook. Log4j is an open source tool developed for putting log statements into our application. Its speed and flexibility allows log statements to remain in shipped code while giving the user the ability to enable logging at runtime without modifying the application binary. The logging behavior can be controlled by editing a configuration file. The property files in Chinook are **log4j.chinook.properties** and **log4j.filewire.properties** under **resources** directory.

2.0.7 Junit

[JUnit](#) is a regression testing framework written by Erich Gamma and Kent Beck. It is used by the developer who implements unit tests in Java. JUnit is Open Source Software, released under the Common Public License Version 1.0 and hosted on SourceForge.

3.0 Setting Up

In this section, we are going to guide you through setting up the Chinook project environment using two different IDEs, Eclipse and JBuilder. We also show you how to run the Chinook p2pNode, the Chinook server, the Chinook Client, and the Perl code in these environments.

3.0.1 Building an Eclipse environment

[Eclipse](#) is an open source project. This manual uses the version 2.1.2. As this manual is written, a new version 3.0 is available. Feel free to try it out.

3.0.1.1 Creating a new project using local files

Step 1. Creating a new project

Start Eclipse, and then click **File** → **New** → **Project...** The following window will appear.

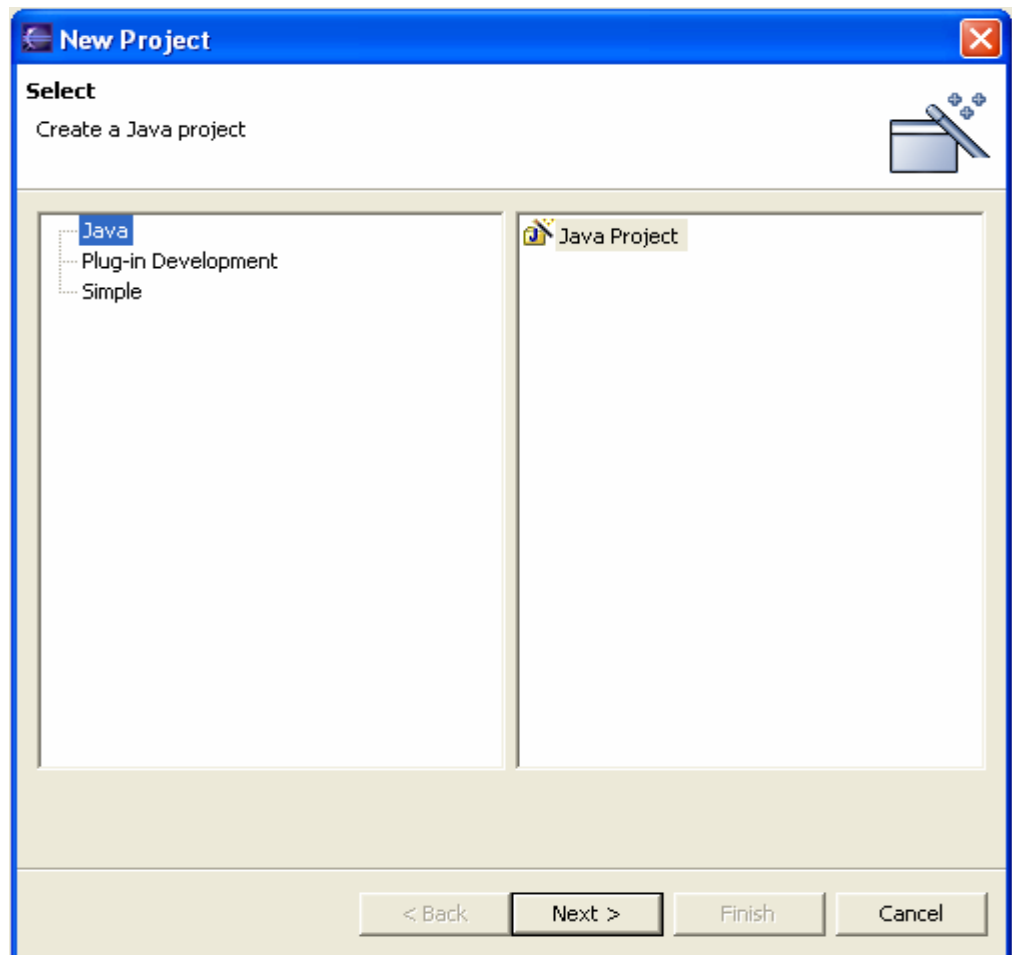


FIGURE 3.1 Eclipse New Project window

Click **Java** in the left pane, **Java Project** in the right pane; then click the **Next** button. The following window will appear.

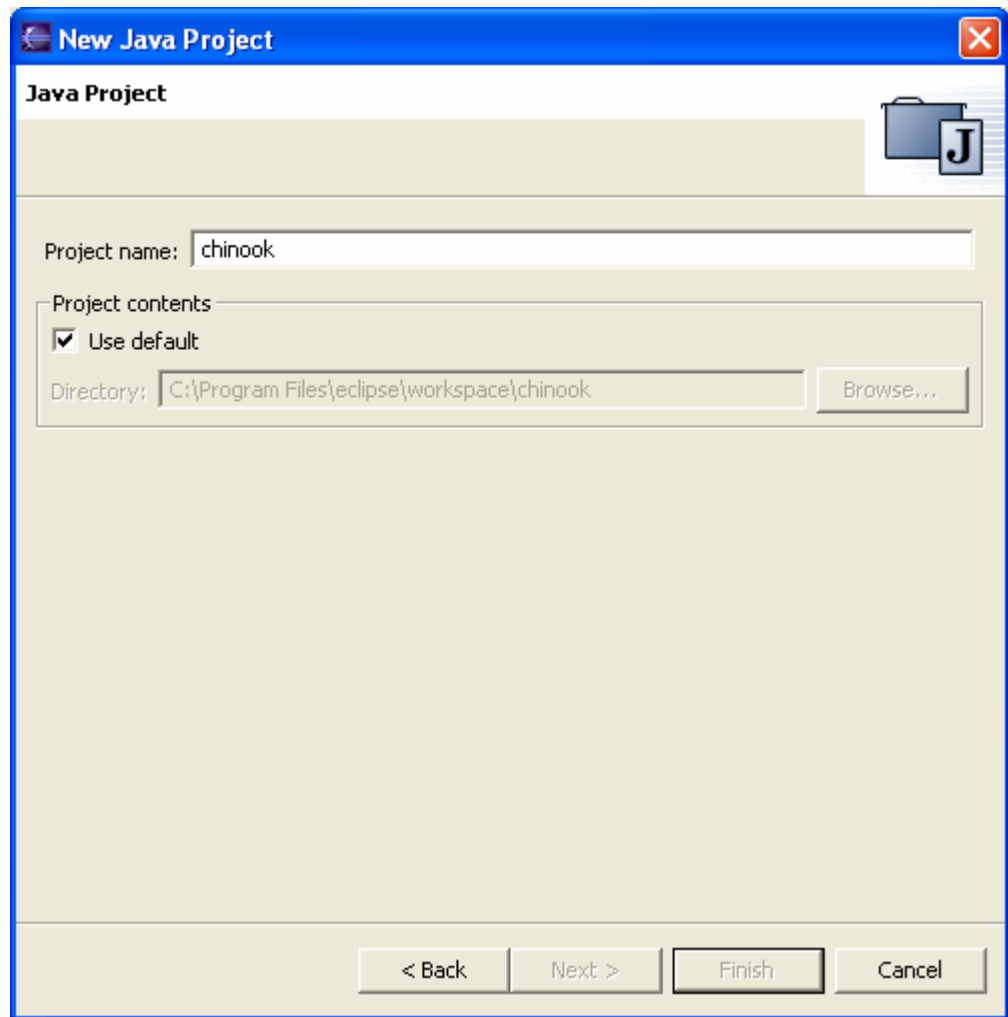


FIGURE 3.2 Eclipse New Java Project window

Enter chinook in the Project Name. You can save the workspace to another place if you don't want the default location. Click the **Finish** button. Click the **Yes** button on **Confirm Perspective Switch**.

A new project chinook has been established in your eclipse workspace. The next step will show you how to import the existing Chinook code into the new created project.

Step 2. Importing code into project

Click the new created project chinook in the **navigator** window; then click **File** → **Import...** The following window will appear.

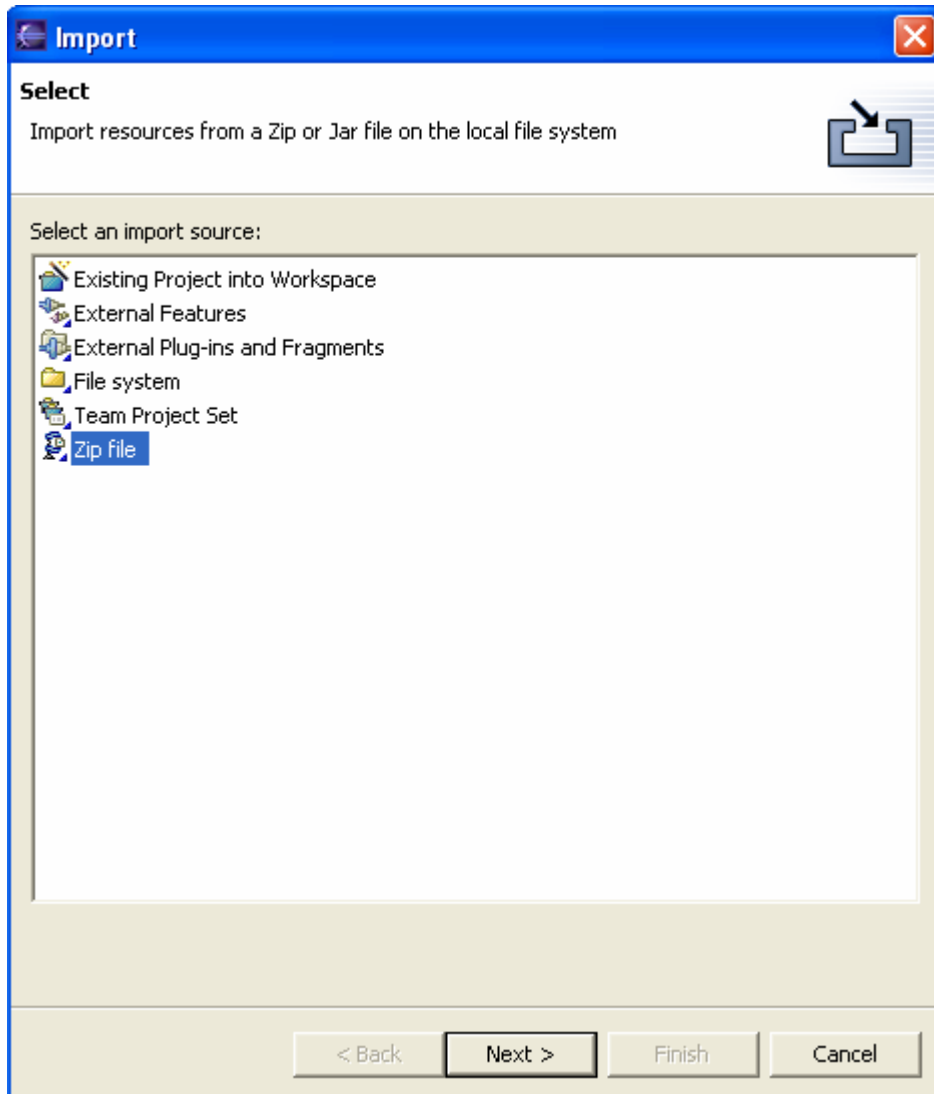


FIGURE 3.3 Eclipse Import window

At this time, you should have already got a working copy of Chinook code, either in zip file or you already unzipped it into your computer. You can import the code from a zip file or from the unzipped files in your file system. We will show you how to import chinook code from file system.

Click **File System** in the panel; and then click the Next button. The following window will appear.

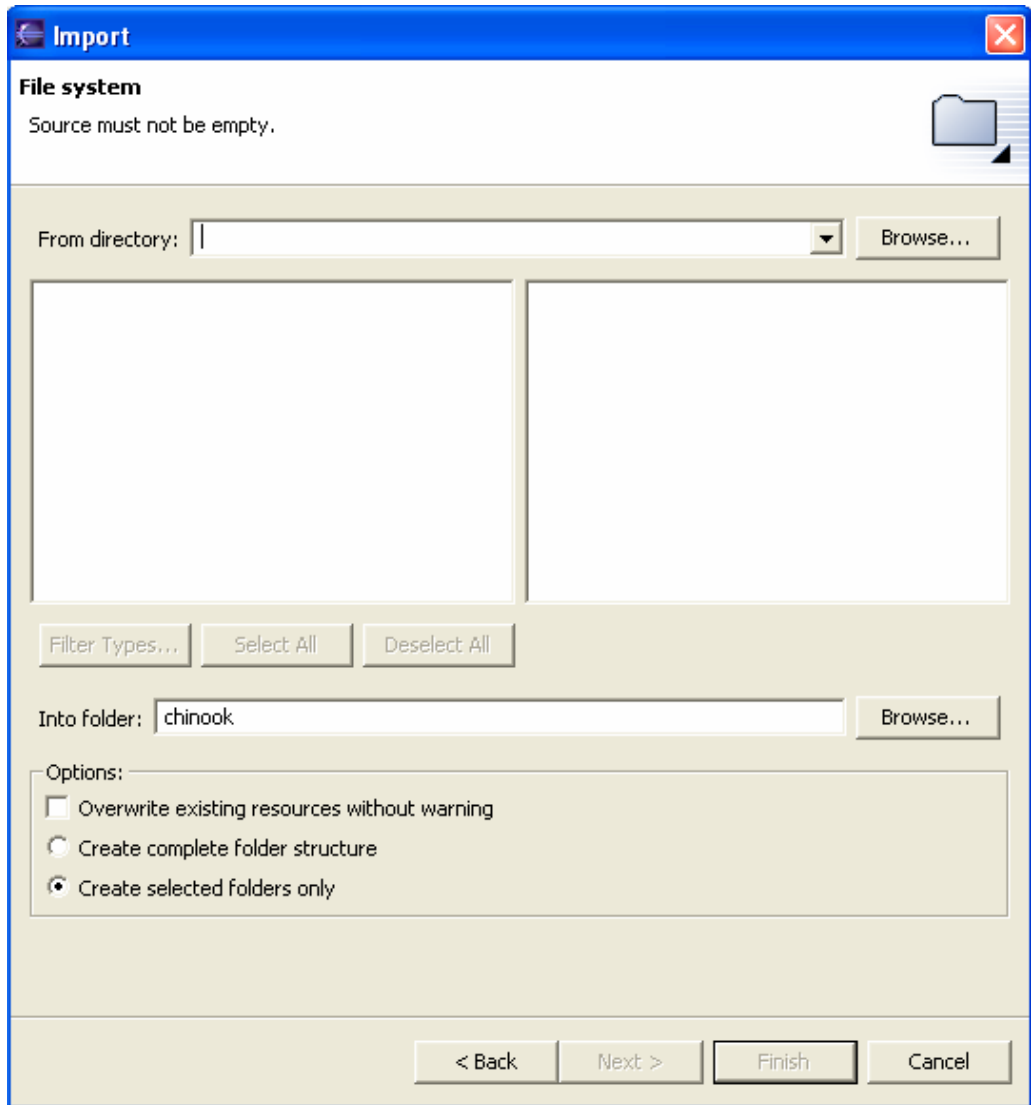


FIGURE 3.3 Eclipse Import next window

Click the **Browse...** button, and then browse to the directory you saved your chinook code. Click the **OK** button to return to the Import window. The following window will appear. Select all the subdirectories and all the files in the right panel. Make sure the **Create selected folders only** is selected. Click the **Finish** button.

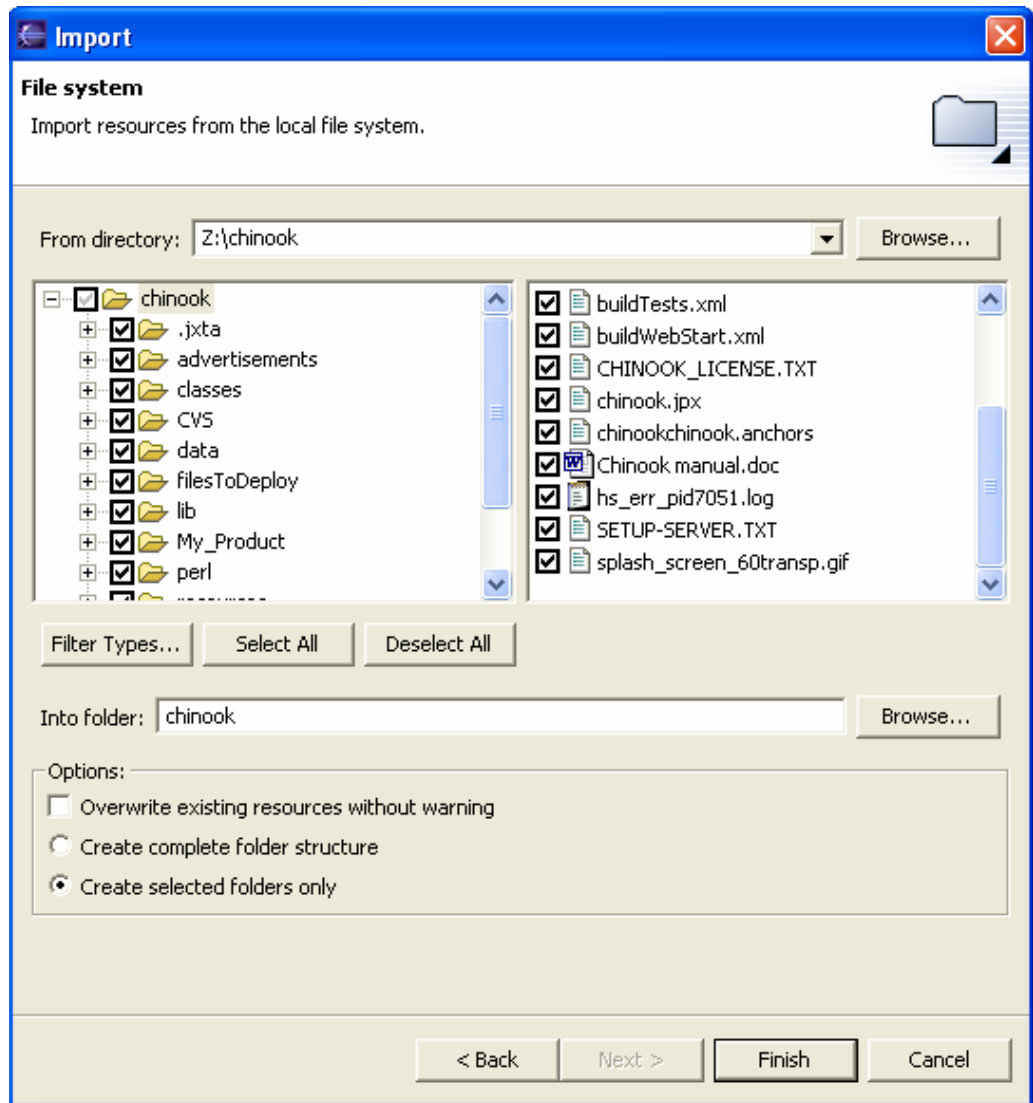


FIGURE 3.4 Eclipse Import finish window

Now you have imported all the files you need for the chinook project. The next step is to setup the environment of the project.

3.0.1.2 Creating a project from CVS

Step 1. Setting up connection to CVS repository.

In order to get the code from CVS repository, you need to setup a connection to the **pserver** containing the code. The following steps assume you are working within the GSC center. If you are working at home, you need to establish a ssh tunnel first before checking out code. Starting Eclipse, then click the CVS repository exploring perspective button on the left bar. If the button is not there, you can go to **Windows-> Open Perspective -> Other....** Select Open CVS Repository Exploring. Right

click in the CVS Repositories perspective window, then select New-> Repository Location... The following window should appear.

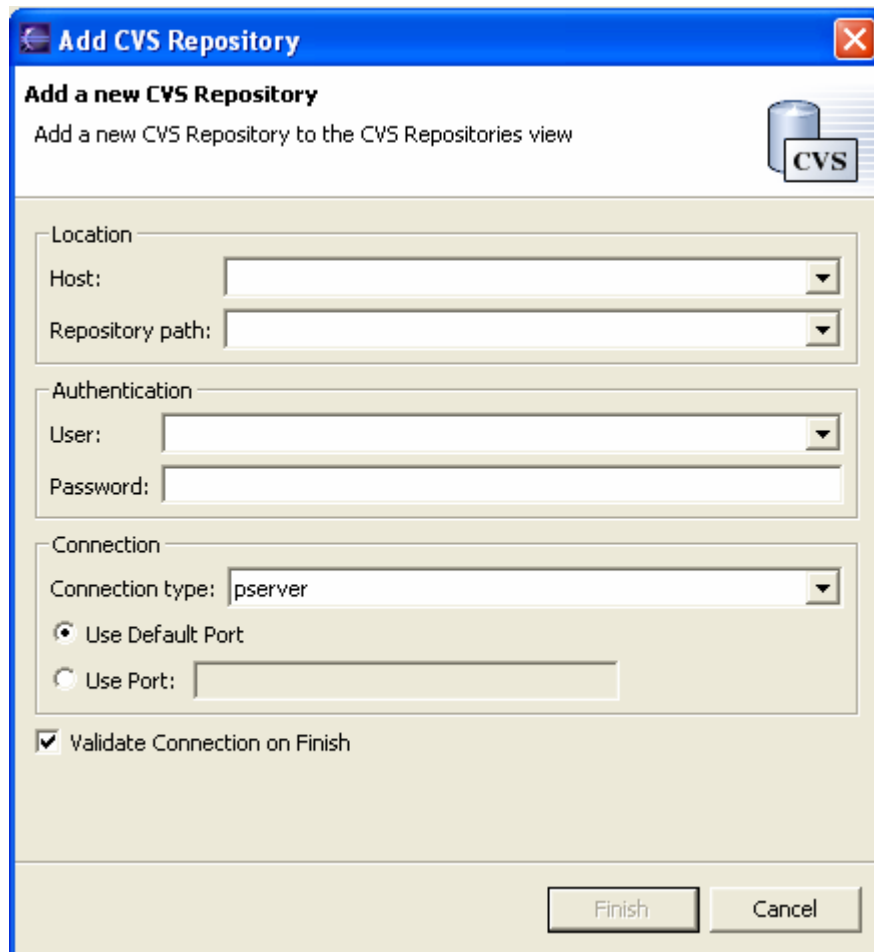


FIGURE 3.5 Eclipse CVS repository window

Entering the information according to the following window. Replace the **User** and **Password** with your own user name and password, then click **Finish** button. Now it has established a connection with the Pserver. The next step will tell you how to check out the code from the repository.

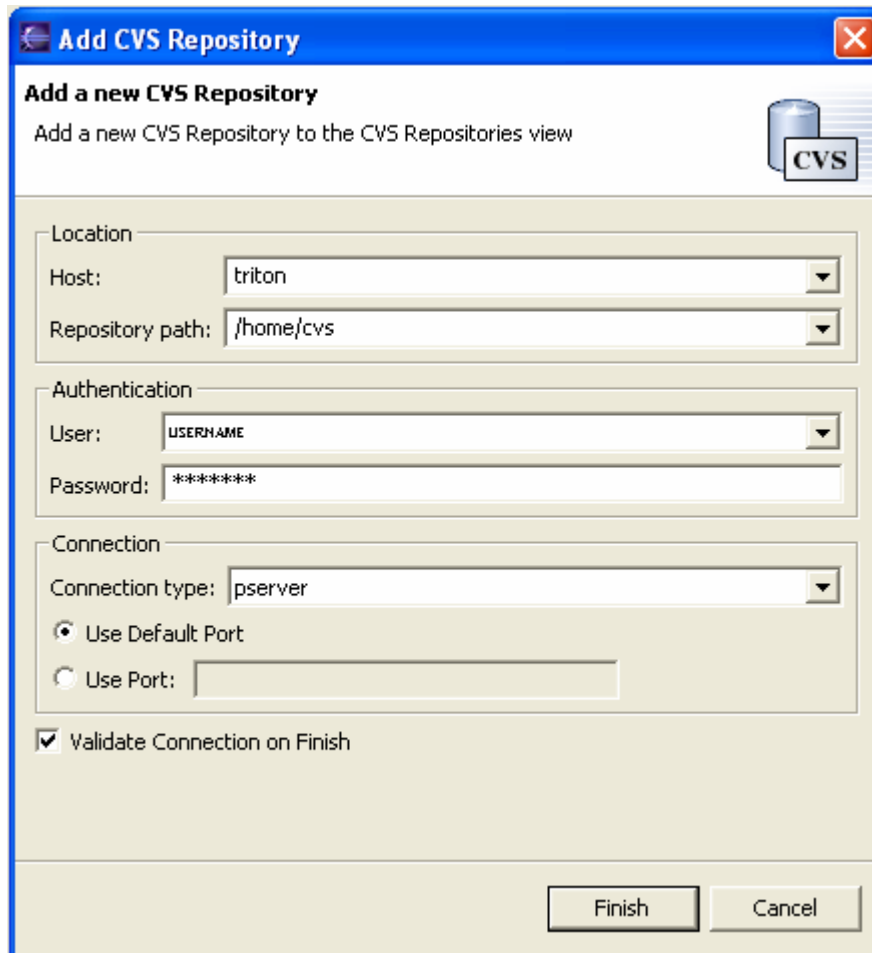


FIGURE 3.6 Eclipse CVS repository window

Step 2 Checking out code from the repository.

Click the plus sign on the left of “pserver:USERNAME@triton:/home/cvs” in the left CVS Repositories perspective to expand the folder, then expand the **HEAD**. Right click the **chinook** Folder, select **Check Out As...** the following window will appear.

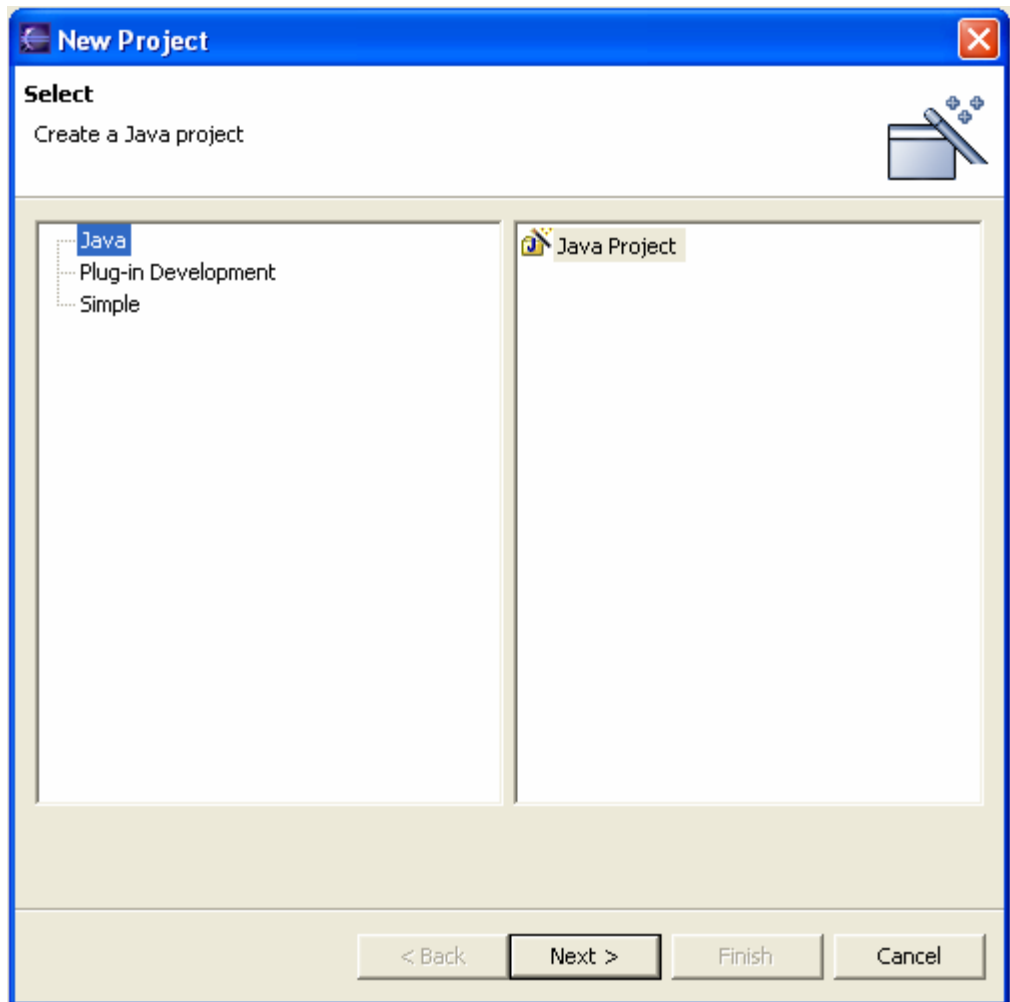


FIGURE 3.7 Eclipse New Project window

Click **Java** in the left pane; Click **Java Project** on the right pane; then click the **Next** button. Enter chinook in the Project Name. You can save the workspace to another place if you don't want the default location. Click **Finish** button. Click **Yes** on **Confirm Perspective Switch**.

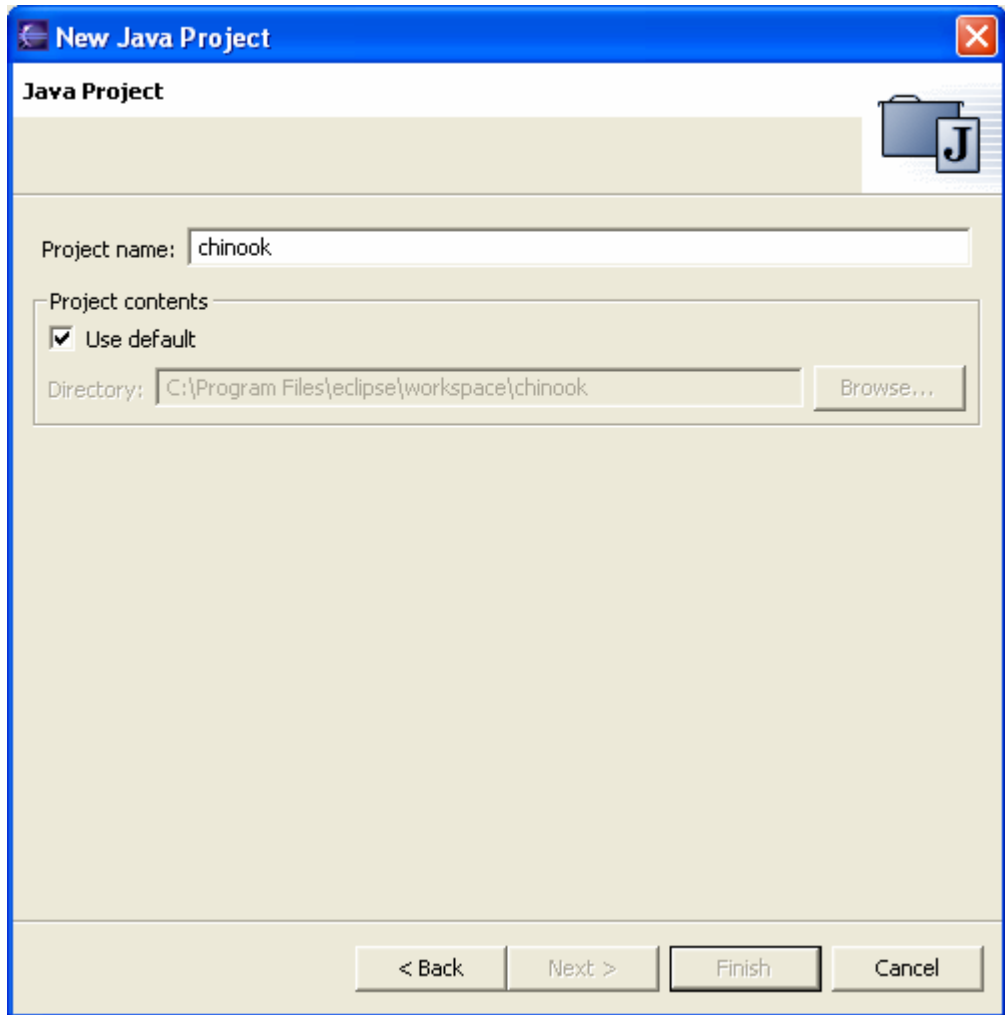


FIGURE 3.8 Eclipse New Java Project window

3.0.1.3 Setup environment

Right click on the project chinook in the **Navigator** window. Select **Properties**. Setup the properties according to the following window. Click **Java Build Path** on the left pane; select chinook on the right pane; then click **Edit...** button. Browse the directory; and then click **src** folder. In the bottom text fields, type in chinook/classes.

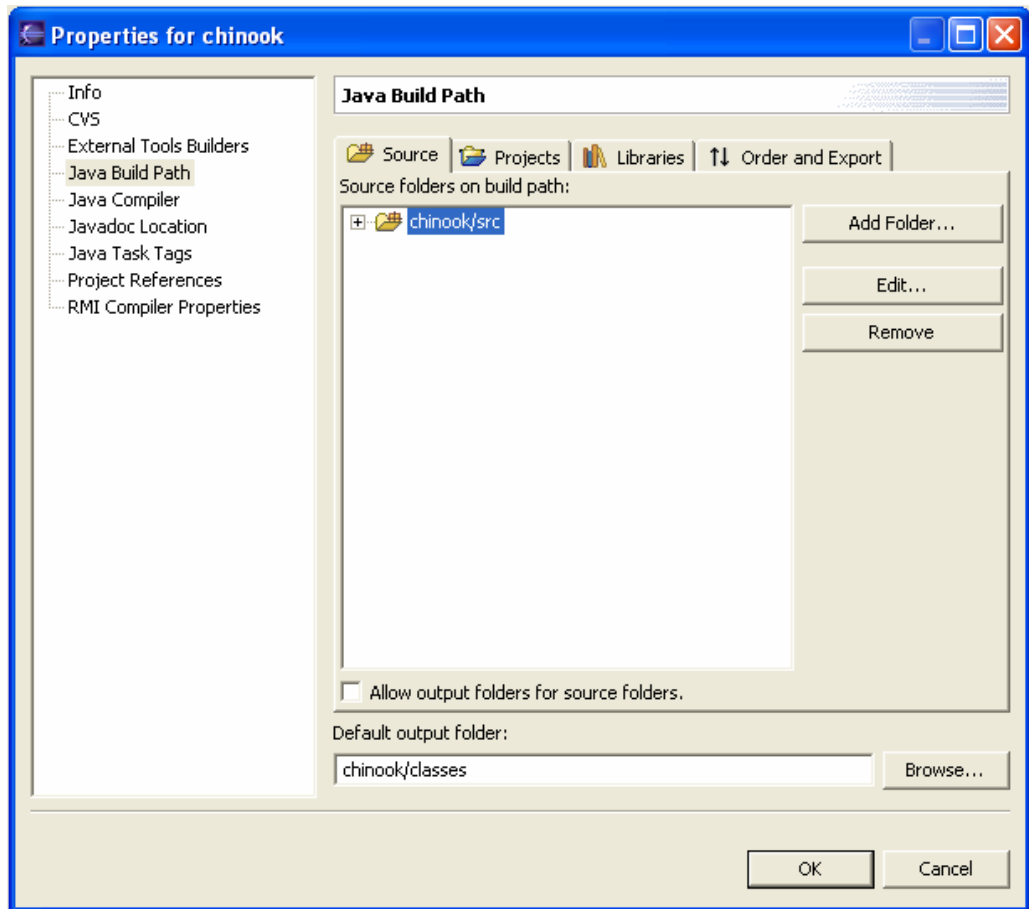


FIGURE 3.9 Eclipse Project Properties window

Click the **Libraries** tab. Click **Add Jars ...** button on the right. Go to **chinook/lib** then select all file under the **lib** folder. Click **OK** button. Click **Add Class Folder...** on the right. select **chinook/resources** folder, then click **OK** button. When prompt **Setting Building Paths** information dialog, click **Yes**.

Now you have successfully set up the project environment.

3.0.2 Building a JBuilder environment

Need to be done

3.0.3 Running a p2p node from an IDE

In the Navigator window, double click to open the file **ChinookP2PNode.java**, which is under the path **chinook/src/ca/bcgsc/chinook/p2p/exec**. Click **Run → Run ...** Double click the **Java Application** on the left pane. The window like the follow will appear.

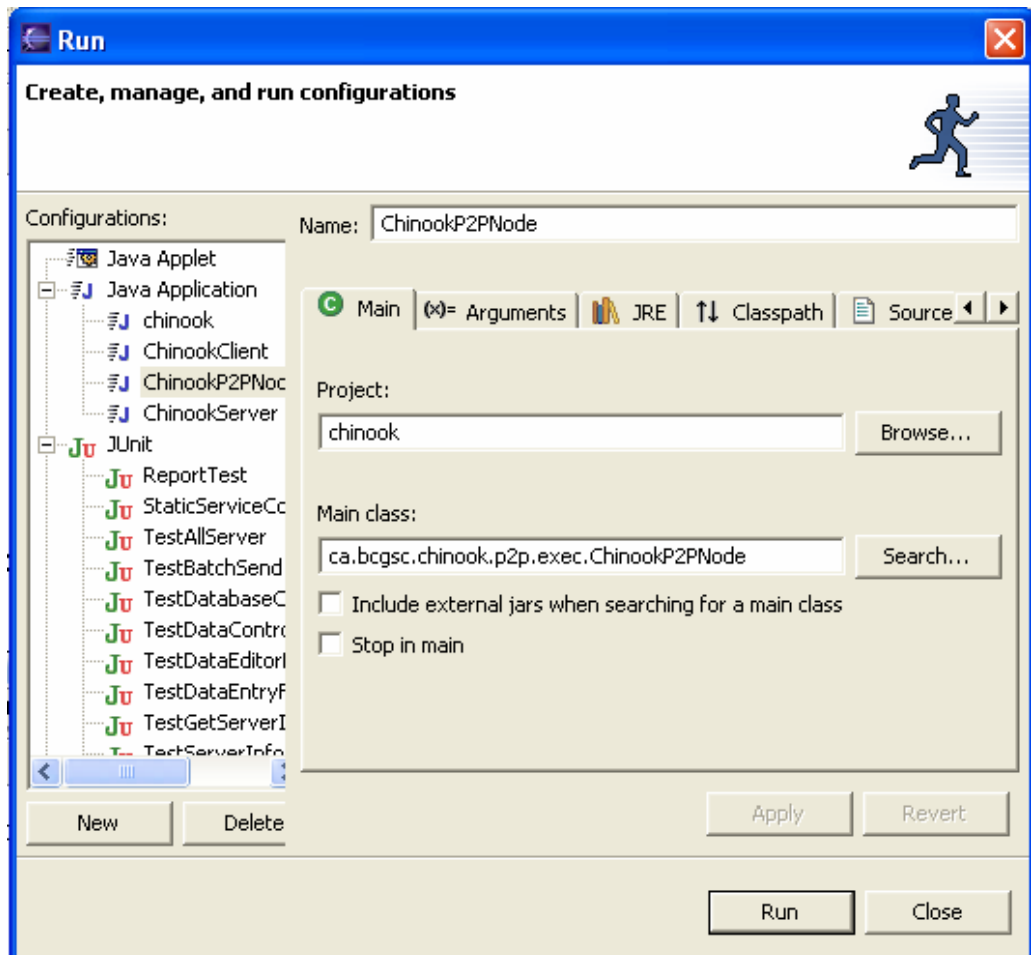


FIGURE 310 Running Chinook p2pNode window

Click the **Run** button. The **p2pNode** will start to run.

3.0.4 Running a server from an IDE

In the Navigator window, double click to open the file `ChinookServer.java`, which under the path `chinook/src/ca/bcgsc/chinook/server/exec`. Click **Run->Run ...** Double click the Java Application on the left pane. The window like the follow will appear.

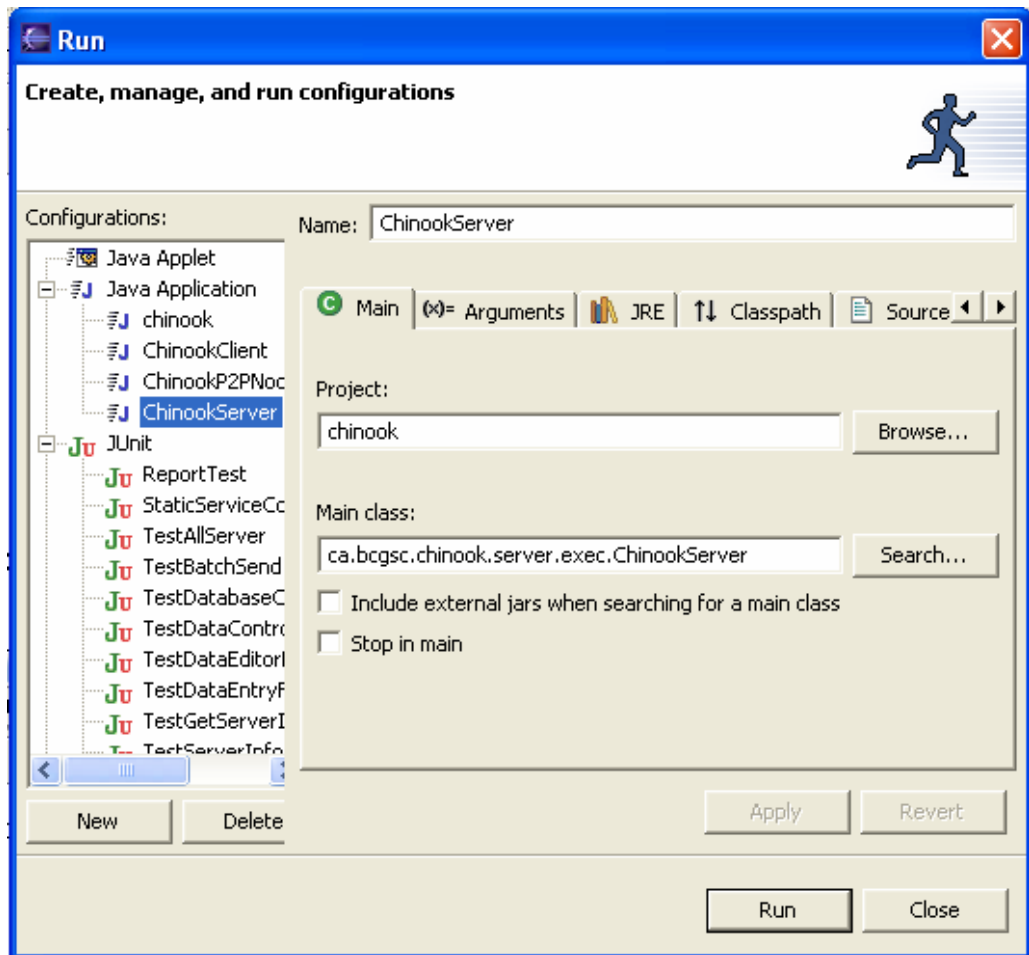


FIGURE 3.11 Running Chinook Server window

Click Arguments tab. You need to specify a few parameters for the server to run. Basically, these are RMI security manager and codebase properties. In the VM arguments, enter information like the following (you need to customize the parameters according to your own Chinook installation):

```
-
Djava.rmi.server.codebase="file:///C:/Program%20Files/eclipse/workspace
/chinook/classes/
file:///C:/Program%20Files/eclipse/workspace/chinook/lib/filewire.jar"
```

```
-
Djava.security.policy="file:///C:/Program%20Files/eclipse/workspace/chinook/resources/chinookRMI.policy"
```

Click Run button, the server will start running.

3.0.5 Running a client from an IDE

In the Navigator window, double click to open the file **ChinookClient.java**, which is under the path **chinook/src/ca/bcgsc/chinook/client/exec**. Click **Run → Run ...**, and then double click the **Java Application** on the left pane. A window similar to the follow will appear.

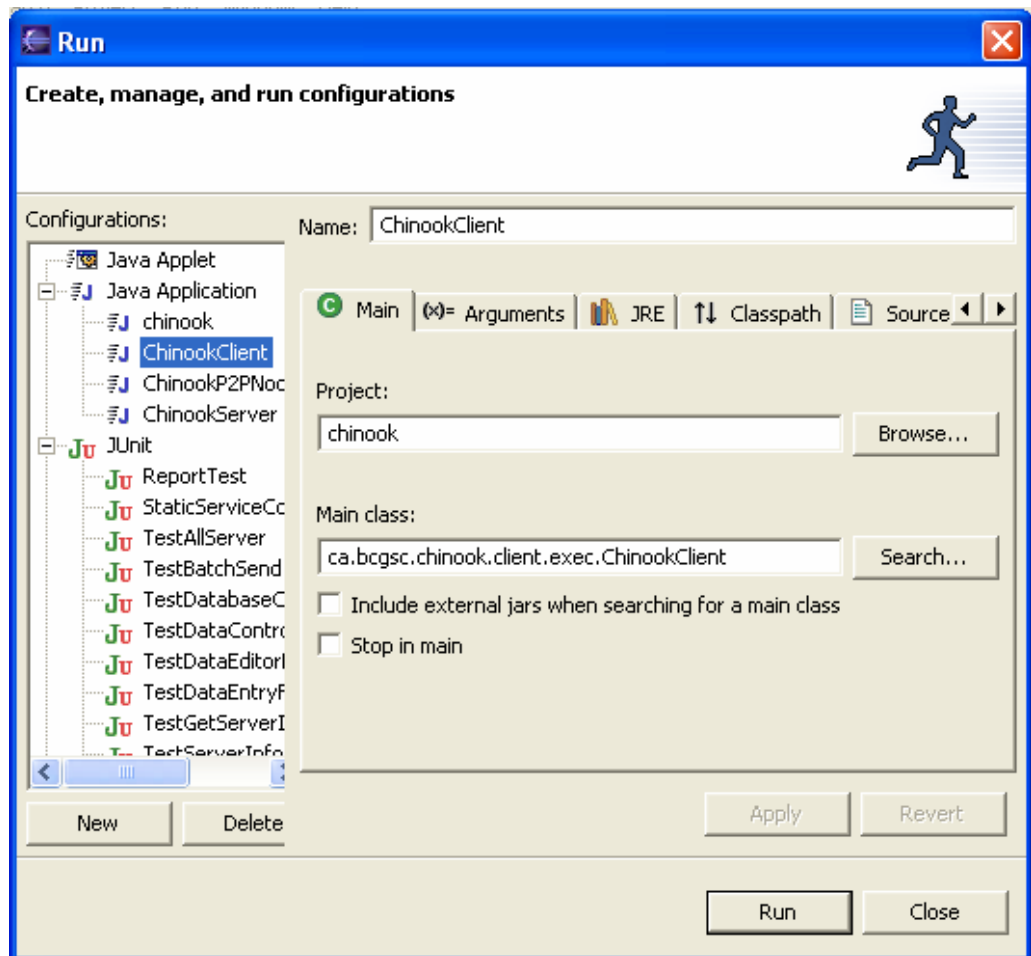


FIGURE 3.12 Running Chinook Client window

Click the **Run** button. The **Client** will start to run.

3.0.6 Running the Perl code from an IDE

Need to be done

4.0 Detailed Architecture

In this section, we are going to give you some detailed package information about the chinook project.

4.0.1 Package Structure

The following section describes the packages made up of Chinook source distribution

4.0.1.1. Client packages

```
ca.bcgs.c.chinook.client --|The client classes
  comobj                --|The serialized objects the client sends to the server
  configuration         --|Configuration classes for data entry and static services
  data                  --|Classes for download result from server and job is being processed
                        by server
  exec                  --|The main class for Client
  gui                   --|Deprecated GUI components
  gui.ds                --|Contains business objects for GUI elements
  gui.ds.container      --|The container objects for various parameter elements
  gui.ds.sequence       --|Validation class for sequence parameter
  local                 --|The client-side application interface (to talk to the server), and job
                        queuing and pulling mechanisms
  manager               --|Client manager classes (client, job, and service manager)
  manager.event         --|The client event classes(client, job, and service event)
  newgui                --|The GUI elements required to run the client
  newgui.dataentry     --|The GUI elements for entering data for services
  newgui.html           --|The GUI elements for displaying original service developer webpage
  newgui.jobs           --|The GUI elements for submitting job
  newgui.jobs.report    --|The GUI elements for displaying result for the client
  newgui.menu           --|The GUI elements for the client menu bar
  newgui.menu.help     --|The GUI elements for the help menu
  newgui.menu.tools    --|The GUI elements for the tools menu
  newgui.parameterentry --|The GUI elements for the parameter entry
  newgui.parameterentry.comboboxmodels --|The combobox model for parameter entry
  newgui.parameterentry.ext --|The parameter editor classes
  newgui.parameterentry.tablemodels --|The tablemodels for parameter entry
  newgui.services      --|The GUI elements for service table
  newgui.servicetype   --|The GUI element for the service type tree and filter panel
  utils                --|Utilities for extracting metadata and loading sequences
```

4.0.1.2. Server packages

```
ca.bcgs.c.chinook.server --|The server classes
  comobj                --|The serialized objects that the sever sends to the client
  configuration         --|The classes to read in the XML data
  configuration.info    --|The classes configure the server-info.xml
  database              --|The Ensembl connection classes
  exec                  --|The main class for starting the server
  gui                   --|The GUI elements for entering new applications
  manager               --|The server manager class
  manager.event         --|The server event and event listener classes
```

remote	--	Remote interfaces and implementations
remote.impl	--	The implementation of the remote interface(RMI/WSDL)
reporting	--	Classes handling report management
runner	--	The parser classes and interface
runner.alignment	--	Parsing classes for alignment services
runner.discover	--	Parsing classes for discover services
runner.elph	--	Parsing classes for elph services
runner.gff	--	Parsing classes for gff services
runner.impl	--	Implementation class for Executable.java
service	--	The service factory class
utils	--	Sequence access and server management utils

4.0.1.3. p2p packages

ca.bcgsc.chinook.p2p	--	The client and server classes required to run Chinook P2P
advertisement	--	The classes for JXTA advertisement
advertisement.configuration	--	The configuration classes for advertisement
client	--	The client classes, discovery management
client.discovery	--	The discovery condition classes
client.event	--	The discovery event classes
configuration	--	Bootstrap configuration classes for Chinook
configuration.ext	--	Configuration classes for client and server
configuration.peergroup	--	The configuration class for Chinook peergroup
data	--	The service cache classes for Chinook
discovery	--	The discovery thread class
exec	--	The main class for p2p node
gui	--	The splash screen for p2p
manager	--	P2p manager class
manager.event	--	p2p event classes
net	--	socket classes for p2p
net.advertisement	--	advertisement socket classes
net.discovery	--	discovery socket classes
server	--	
utils	--	Various Chinook JXTA utils (Discovery and peer groups)

4.0.1.4. batching packages

ca.bcgsc.chinook.batching	--
batch	--
cache	--
cache.impl	--
configuration	--
constants	--
directory	--
directory.impl	--
manager	--
manager.event	--
net	--
net.protocol	--
queue	--
reporting	--

4.0.1.5. parsing packages

ca.bcgsc.chinook.parsing	--
sequence	--

```
setoutput      --|
setoutput.impl --|
```

4.0.1.6. common packages

```
ca.bcgsc.chinook.common--|
  dataentry      --|
  dataentry.configuration --|
  dataentry.guisupport --|
  dataentry.guisupport.impl --|
  dataentry.loader --|
  dataentry.loader.impl --|
  dataentry.objects --|
  dataentry.objects.impl --|
  dataentry.utilities --|
  dataentry.validation --|
  dataentry.validation.impl --|
```

4.0.2 Managers

4.0.2.1 Batch Manager

`ca.bcgac.chinook.batching.manager.BatchManager.java`

4.0.2.2 Client Manager

`ca.bcgsc.chinook.client.manager.ClientManager.java`. The client manager is used to manage the Chinook Client starting and stopping, as well as the listener and event of the Chinook Client.

4.0.2.3 Job Manager

`ca.bcgsc.chinook.client.manager.JobManager.java`. The job manager is used to manage the jobs. It manages the job starting and completing, the listeners and events.

4.0.2.4 Service Manager

`ca.bcgsc.chinook.client.manager.ServiceManager.java`. This manager is used to manage the services. It is used to add, remove, and update services. It also manages the listener and events related to services.

4.0.2.5 Discovery Manager

`ca.bcgsc.chinook.p2p.client.DiscoveryManager.java`. The discovery manager handles the changes in discovery criterion. For example, when a user clicking a particular type in the service type tree, this manager fires an appropriate events to handle refining the search.

4.0.2.6 P2P Manager

ca.bcgsc.chinook.p2p.manager.P2PManager.java. The p2p manager is used to manage p2p events and listeners.

4.0.2.2 Server Manager

ca.bcgsc.chinook.server.manager.ServerManager.java. The server manager is used to manage server events and listeners.

4.0.3 Network (sockets)

Need to be done

4.0.4 XML

XML is used throughout Chinook project. All the configuration files under resources directory and the build file for Apache Ant require you to have knowledge of XML. If you have never learnt XML before, you might find [W3Schools](http://www.w3schools.com) is useful.

4.0.5 Database

Need to be done

4.0.6 GUI

The GUI part of Chinook is mainly in Chinook Client, although a GUI component to facilitate adding new services to Server is under implementation. All the GUI packages are under **ca.bcgsc.chinook.client.newgui** package. Basically, there are seven packages for the GUI functionality. The package **html** is used for the light-weight browser located at the right-bottom corner of the Chinook Client. The **job** package containing the GUI components is used to display job status panel, which is located at the left-bottom corner of the Chinook Client. The **menu** package is used for the Chinook menubar, which is on the top of the Chinook Client. The **services** package is used for displaying service panel, which is located at the right-upper corner of the Chinook Client. The classes under **servicetype** package are used to display the service type tree and filter panel, which are located at the left-upper corner of the Chinook Client. The **dataentry** package provides GUI functionality for entering data to submitted jobs. The **parameterentry** package provides GUI functionality for entering parameters to submitted jobs.

4.0.7 RMI/Web Services

Need to be done

4.0.8 Perl Batching

Need to be done

4.0.9 Filewire

Need to be done

4.0.10 Logging

As said before, Chinook project uses log4j as the logging tool.

```
public class Logging {

    private static final Logger log =
    Logger.getLogger(Logging.class);
    static {
        try {
            URL log4j_resource = Logging.class.
                getClassLoader().getResource(
                    "log4j.chinook.properties");
            if (log4j_resource == null) {
                throw new Exception("Could not load log4j
configuration");
            }
            PropertyConfigurator.configure(log4j_resource);
        }
        catch (Exception e) {
            BasicConfigurator.configure();
            log.warn(
                "Could NOT load
\"log4j.chinook.properties\" - Make sure it is in the
CLASSPATH",
                e);
        }
    }

    public static void enable() {
        //does nothing but ensure that the log resource has
        been loaded via the class
        //loader.
    }
}
```

4.0.11 Junit

Need to be done

5.0 Walkthrough

5.0.1 Extending the GUI

Need to be done

5.0.2 Adding new PERL/JAVA commands

Need to be done

5.0.3 Adding a new Database

Need to be done

5.0.4 Adding a new Data Entry Object Type

Need to be done

5.0.5 Deploying Chinook over Java Web Start

Java Web Start (<http://java.sun.com/products/javawebstart/>) allows the Chinook application to be deployed over a network with only a single-click from a Java-enabled Internet browser. Instead of requiring users to download and install Chinook, we provide this mode of executing Chinook clients as it fits with our tenet of reducing the complexity of use of bioinformatics applications.

This walkthrough gives step-by-step instructions as to how Chinook is deployed over Java Web Start.

Follow the steps:

1. Install Tomcat (follow instructions at <http://jakarta.apache.org/tomcat/>)
2. Ensure that the JNLP mime-mapping has been installed into the `web.xml` file in the `conf/` folder of your Tomcat installation. If it is not there, add the following to your `web.xml` file:

```
<mime-mapping>
  <extension>jnlp</extension>
  <mime-type>application/x-java-jnlp-file</mime-type>
</mime-mapping>
```

3. Go to the `webapps/` folder of your Tomcat installation. Create a new directory called `webstart/`. Go to the `webstart/` folder and now create a new folder called `chinook-client/`. This is where we will place all the Java Web Start files when we deploy Chinook.
4. In the `webapps/webstart/` folder under your Tomcat installation. Create a directory called `WEB-INF/`. In this directory copy an existing `web.xml` file from another Tomcat `WEB-INF/` directory to this one (I took one from the `webapps/tomcat-docs/WEB-INF` directory). Edit this file adding both a `display-name` and a `description` (can be whatever you want to add for both).
5. **Setting up Ant to Deploy Chinook to Java Web Start.** To deploy the Chinook code to our Tomcat server, we use an Ant script called `buildWebStart.xml`. To configure this Ant script, we first need to point to the correct directory for the installation. Open the `buildPersonal.properties` file in your Chinook directory. Set the value of the `WEBSTART_DEPLOY_DIR` to the absolute location of the `chinook-client/` folder that you created in Step 3. Then run the `build.xml` ant script to build the `chinook.jar` jar file. To do this from the command line, type the following:

```
ant build
```

6. **Configuring the JNLP file.** Before you deploy Chinook. You will need to configure the JNLP file for your server location. Otherwise, you will be deploying and pointing to the default web start (probably the one at the BCGSC). Go to the `webstart/` directory in your Chinook folder and open the `ChinookClientLauncher.jnlp` file. The top tag called `<jnlp>` will likely be pointing to a URL other than your own. Change the bold characters in the text below to point to your Tomcat URL.

```
<jnlp spec="1.0+"  
  codebase="http://localhost:8080/webstart/chinook-  
  client" href="ChinookClientLauncher.jnlp">
```

7. **Deploying Chinook using Ant.** To deploy Chinook using the `buildWebStart.xml` file, first run the `genkey` target (you only ever need to run this once). This will generate the keys for signing your jar files.

```
ant -f buildWebStart.xml genkey
```

Then run the `build` target. This will sign the jars and put them in the `signedJars/` directory of your Chinook folder. (It will create one if it doesn't exist)

```
ant -f buildWebStart.xml build
```

Finally, run the deploy target. This will copy over the necessary files to Tomcat.

```
ant -f buildWebStart.xml deploy
```

8. **Problems deploying.** Ensure that the Internet is connected. Ensure that all the jars have been signed. If you are developing Chinook and add new jars, these need to be referenced in the JNLP file and in the sign target of the buildWebStart.xml Ant script.
9. **Congratulations.** You should now be able to run Chinook by going to the ChinookClientLauncher.html page in your Tomcat installation. It should be located at `http://mytomcat:8080/webstart/chinook-client/ChinookClientLauncher.html` where **mytomcat** is the name of your Tomcat machine. If you have any problems with this walkthrough, please contact us (see Section [6.0.1 Mailing List](#)).

5.0.6 Adding a new Junit test

Junit framework enables you to write repeatable tests. In this section, we will guide you through how to write a Junit test. For example, we implemented a Junit class used to test **DatabaseController** class.

```
import junit.framework.TestCase;
import junit.framework.TestSuite;

import ca.bcgsc.chinook.junit.TestConstants;
import ca.bcgsc.chinook.logging.Logging;
import ca.bcgsc.chinook.server.database.DatabaseTypeController;

import org.apache.log4j.Logger;

public class TestDatabaseController
    extends TestCase {
    public static final String TEST_ALL_TEST_TYPE = TestConstants.
        TEST_TYPE_SERVER;

    private static final Logger log =
        Logger.getLogger(TestDatabaseController.class);

    static {
        Logging.enable();
    }

    public TestDatabaseController(String s) {
        super(s);
    }
}
```

```

/**
 * Automatically called by the JUnit test framework prior to every
 * test case being run.
 */
protected void setUp() throws Exception {
    super.setUp();
}

/**TESTS**/
public void testGetDatabaseController() {
    String[] datadata = DatabaseTypeController.getInstance().
        getAvailableDatabaseData();
    Assert.assertNotNull(datadata);
    for (int i = 0; i < datadata.length; i++) {
        log.info("DATA_DATA REGISTERED: " + datadata[i]);
    }

    String database_data = datadata[0];

    String[] database_types = DatabaseTypeController.getInstance().
        getAvailableDatabaseTypes(database_data);
    Assert.assertNotNull(database_types);
    for (int i = 0; i < database_types.length; i++) {
        log.info("DATABASE_TYPES REGISTERED: " + database_types[i] +
            " FOR DATABASE_DATA: " + database_data);
    }

    String database_type = database_types[0];

    String[] species =
        DatabaseTypeController.getInstance().getAvailableSpecies(
            database_data, database_type);
    Assert.assertNotNull(species);
    for (int i = 0; i < species.length; i++) {
        log.info("SPECIES REGISTERED: " + species[i] +
            " FOR DATABASE_DATA: " + database_data +
            " FOR DATABASE_TYPE: " + database_type);
    }

    String spec = species[0];

    String[] databases = DatabaseTypeController.getInstance().
        getAvailableDatabases(
            database_data, database_type, spec);
    Assert.assertNotNull(databases);
    for (int i = 0; i < databases.length; i++) {
        log.info("DATABASE REGISTERED: " + databases[i] +
            " FOR DATABASE_DATA: " + database_data +
            " FOR DATABASE_TYPE: " + database_type +
            " FOR SPECIES: " + spec);
    }
}

// overrides superclass method: called after every test method
protected void tearDown() throws Exception {
    super.tearDown();
}

/**
 * Factory method to return a test suite consisting of all

```

```

    * test operations in this unit test.
    * @return A JUnit TestSuite object containing instances of all
    *         tests to execute.
    */
    public static TestSuite suite() {
        TestSuite testSuite = new
TestSuite(TestDatabaseController.class);
        return testSuite;
    }
}

```

```
public class TestDatabaseController extends TestCase
```

This line defines the Junit test class whose name is `TestDatabaseController`. All Junit test classes are subclasses of `TestCase` class. You need to extend `TestCase` class to implement a Junit test.

```
public static final String TEST_ALL_TEST_TYPE =
TestConstants.TEST_TYPE_SERVER;
```

these lines are used to specify what kind of test type is this Junit test. It can be one of the following five types. `TEST_TYPE_SERVER`, `TEST_TYPE_CLIENT`, `TEST_TYPE_BATCHING`, `TEST_TYPE_PARSING`, `TEST_USER_INPUT_REQUIRED`. We can use this type information to test the server or client etc. as a whole unit.

```
private static final Logger log =
Logger.getLogger(TestDatabaseController.class);

static {
    Logging.enable();
}

```

These lines enable the logging in Chinook, so we can use `warn`, `info`, `error` etc. methods to log Chinook.

```
protected void setUp() throws Exception {
    super.setUp();
}

```

These lines are used to setup common fixtures. In Junit class, if you have more than one test cases, and you have some common fixtures for running these tests, you can use `setUp()` method to initialize the fixture's objects. The symmetric operation to `setUp()` is `tearDown()`, which is used to clean up the test fixture at the end of a test. Each test runs in its own fixture. Junit calls `setUp` and `tearDown` for each test so that there can be no side effects among test runs.

```
public void testGetDatabaseController() {
    . . .
}

```

These lines define the concrete test cases. You can have as many test cases as you like.

```
public static TestSuite suite() {
```

```
    TestSuite testSuite = new TestSuite(TestDatabaseController.class);  
    return testSuite;  
}
```

This static method allows use to run all the test cases together.

6.0 Further Information

Chinook is funded by Genome Canada as part of the Bioinformatics of Mammalian Gene Regulation grant. Stephen Montgomery is a Ph.D. graduate student in Genetics at the University of British Columbia. He is funded by the Michael Smith Foundation for Health Research. His work is performed primarily at Canada's Michael Smith Genome Sciences Centre as part of the Gene Regulation Informatics team. Steven Jones is the Head of Bioinformatics for the CMSGSC. He is a Scholar of the Michael Smith Foundation for Health Research. Currently, there is no source code available for Chinook. The source though is licensed under Creative Common's Attribution-Non-Commercial license and is freely available on request to chinook@bcgsc.bc.ca. If you have any questions and find any bugs in Chinook, please email us.

6.0.1 Mailing List

The Chinook mailing list is a low-volume regulated list that broadcasts weekly development announcements. We recommend you to sign up the mailing list at <http://www.bcgsc.ca/mailman/listinfo/chinook>. You will get the latest information about Chinook (including upgrade, bug fix). You can also view the Archive at <http://www.bcgsc.ca/pipermail/chinook/>.

[Back to Table of Contents](#)

6.0.2 Authors

Montgomery SB, Fu T, Guan J, Jones SJM (in preparation).

Chinook Internal:

Chinook Service Developers:

Monica Sleumer, Keven Lin, Tamara Astakhova, Jun Guan, Maik Hassel, James Kennedy, Eddy Tsang, Yvonne Li, Tony Fu

Other thanks:

Asim Siddiqui, Misha Bilenky, Gordon Robertson

Chinook External:

Jonathan Lim, Wyeth Wasserman, David He, Sohrab Shah, Francis Ouellette

[Back to Table of Contents](#)

6.0.3 Known Problems

1. Web pages may not be displayed properly in the **Lightweight Web Browser**. This is because Chinook is a Java application, and JEditorPane, which is used to display web pages, only supports Html 3.2 currently. Any web pages created using html version above 3.2 will likely not be displayed properly.

[Back to Table of Contents](#)

6.0.4 License



SOME RIGHTS RESERVED Chinook is licensed under a [Creative Commons License](#).

[Back to Table of Contents](#)