OpenJVS PROGRAMMER'S GUIDE

Developing Custom Solutions With OpenLink's OpenJVS

1502 RexCorp Plaza

15th Floor, West Tower

Uniondale, NY 11556

TEL: (516) 227-6600 FAX: (516) 227-1799

WEBSITE: http://www.olf.com

REVISION HISTORY

Document Name	Date	Description	Version
Open JVS Programmer's Guide	12/22/2008	New	V9.0r1
	3/31/10	Update	V9.0r1 and up

Trademarks

OpenLink® and Endur® are registered trademarks of OpenLink Financial, Inc. Findur™, Toolkit™, Connex™, *pMotion*™, and *gMotion*™ are trademarks of OpenLink Financial, Inc.

All products, company names and marks mentioned in this document are the property of their respective owners and are mentioned for identification purposes only.

Other product names mentioned in this document may be trademarks or registered trademarks of their respective companies and are hereby acknowledged.

TABLE OF CONTENTS

1.0 OpenJVS OVERVIEW

The OpenJVS Programmer's Guide provides guidelines for setting up the OpenJVS development environment and step-by-step walkthroughs for developing OpenJVS Plugins.

1.1 What is OpenJVS?

OpenJVS provides a platform for client customizations of OpenLink's product and allows the development of rich Java business logic with OpenJVS as the base.

1.2 What are the Capabilities of OpenJVS?

OpenJVS functionality includes, but is not limited to, the following:

Functionality	Description
Reporting	Running standard reports as well as creating customized reports viewable and printable from the OpenLink Report Viewer.
Automation	Automating OpenLink System processes such as the generation of confirmations by the back-office or other End-of-Day processing.
Integration	Communicating with third-party software such as Crystal Reports, Lotus Notes and legacy systems.
Custom Analytics	Providing the ability to develop user defined result types for risk evaluation and simulations ("what if scenarios").

1.3 How OpenJVS Works

Plugins and Projects

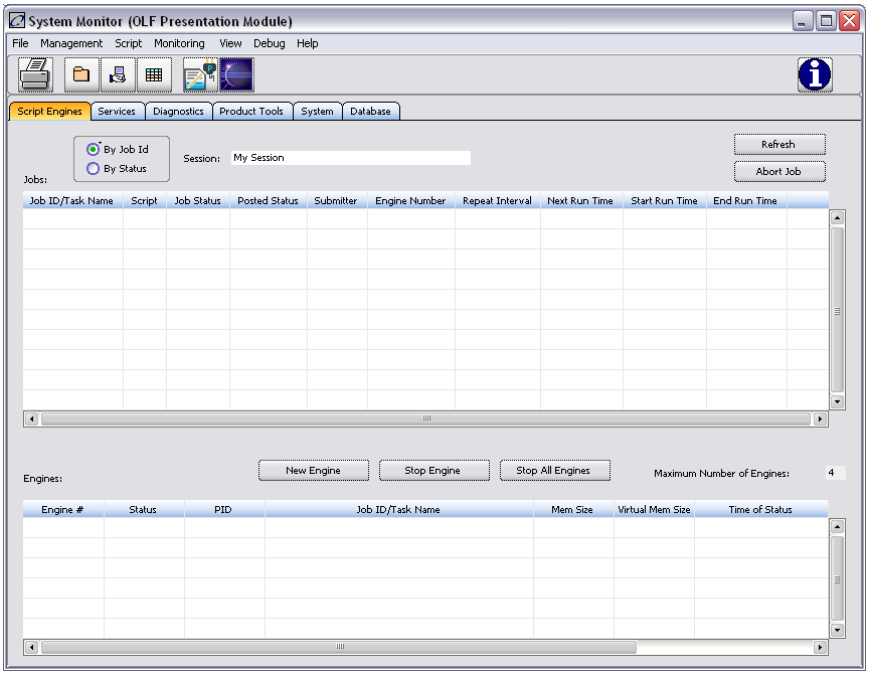
OpenJVS is a Java based scripting language that works in conjunction with the OpenLink System. Custom plugins (formerly called scripts) can be created, developed, debugged and maintained using a standard Java IDE outside the OpenLink System. Projects can also be created, imported, exported, and deleted outside the OpenLink System.

OpenLink Script Engine (SCRENG)

The OpenLink Script Engine (SCRENG) is an independent process that will start the JVM (Java Virtual Machine). The Java source code is compiled by the Java compiler and saved into the OpenLink System during a "Save Project" or "Save Plugin" request. Both the source code and bytecode are saved. Bytecode is the form of instructions that the JVM executes. At runtime, the OpenLink System loads the bytecode from the database into the JVM.

A running plugin is referred to as a "job," while multiple jobs are batched in a "job queue". If there is a SCRENG running, it runs the next job. Any number of script engines can be run, up to the number specified by the OpenLink MAX_TASK_ENGINE environment variable. If there is no script engine currently active, one starts automatically when the plugin is run.

Scheduled jobs can be monitored by clicking on the **Script Engines** tab of the **System Monitor**. From this window, script engines can be started and jobs can be terminated.



Tasks

Tasks are used to organize the execution of plugins. Plugins can be saved as a script type that can be logically ordered into a task. There are four script types, described below, that will be discussed in greater detail in later sections of this guide:

Script Type	Description
Parameter Script	Used to gather information that is passed to a main script in a task.
Main Script	Contains the bulk of the process and/or reporting code. A main script can be designated to run alone in a task.
Page Viewer Script	Typically used to set subscriptions and activate the Page Viewer.
Output Script	Run locally after the main script has run, finalizing the outputs.
Include Script	Used to include functionality from other main scripts. This can be used much like the include statement in C-based languages.

1.4 Additional Resources

OpenJVS Javadocs are bundled with OpenLink as a jar file. The *OpenJVS IDE Integration* section of this document provides instructions for adding the OpenJVS Javadocs.

1.5 Prerequisites

This document assumes that the user has a basic understanding of the following:

- The OpenLink System
- The data model within the OpenLink System
- Java.

2.0 OpenJVS TOOLS

The tools used to develop OpenJVS plugins include the following:

- Standard Java IDE
- OpenJVS Java Documentation
- OpenLink System Monitor
- OpenLink AdHoc Query Viewer
- OpenLink Olisten Console.

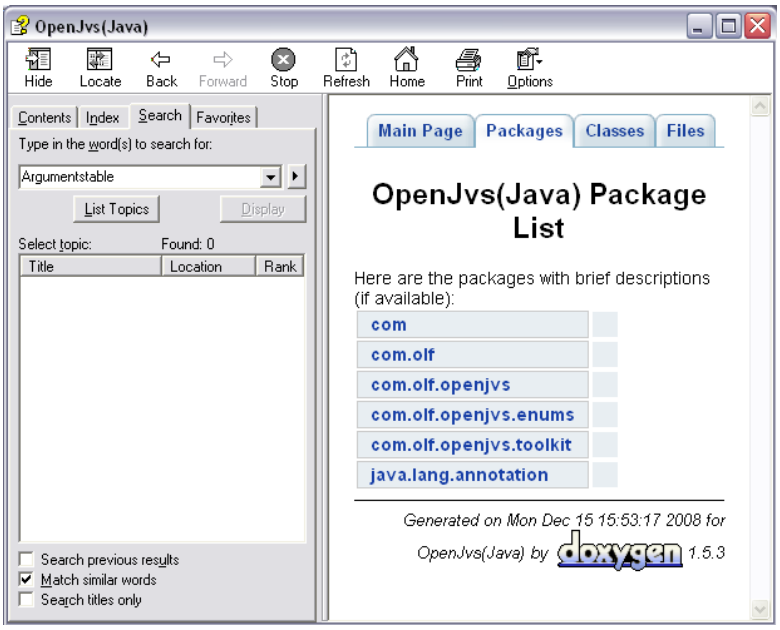
2.1 Standard Java IDE

Custom plugins can be developed and debugged outside the OpenLink System using a standard Java IDE GUI/Script Editor. A locally running session of the OpenLink System must be available. The following free, open-source, Java IDE's are supported for OpenJVS:

Eclipse	Eclipse 3.1.2 can be downloaded and installed from the following location: http://archive.eclipse.org/eclipse/downloads/
NetBeans	NetBeans 5.5.1 can be downloaded and installed from the following location: http://www.netbeans.info/downloads/all.php?b_id=3095

2.2 OpenJVS Java Documentation

The OpenJVS JAVA Documentation, (javadoc) contains the definitions for all of the OpenJVS classes and their corresponding methods.



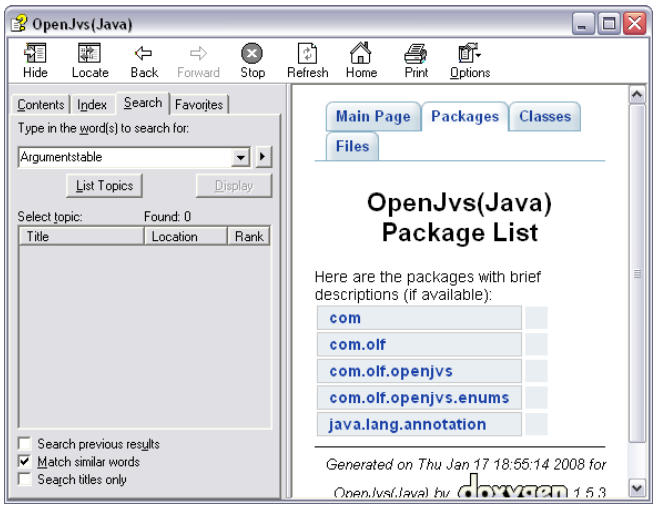
Doxygen is a documentation generator for Java. It extracts documentation from source file comments and is used to create the glossary of OpenJVS classes and methods.

2.2.1 Package List

The **Packages** tab of the javadoc is a list of the different packages included in OpenJVS. Java packages are a mechanism for organizing Java classes into namespaces.

Java packages can be stored in JAR (Java ARchive) compressed files, allowing classes to download faster as a group rather than one at a time.

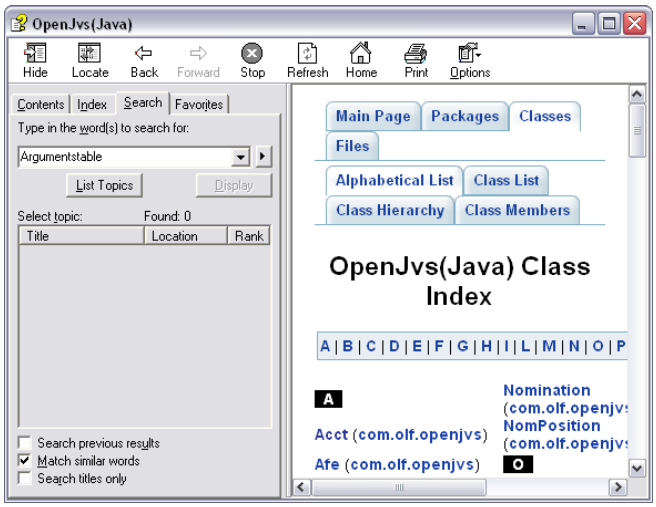
Packages are typically used to organize classes belonging to the same category or providing similar methodology.



2.2.2 Class Index

The **Classes** tab of the javadoc is a list of the different classes within OpenJVS. A class is a programming language construct that is used as a blueprint to create objects. This blueprint includes the attributes and methods that all objects share.

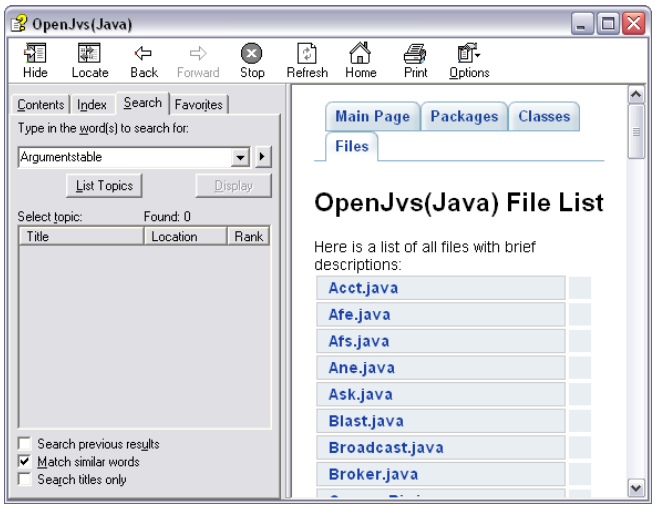
In examples that follow in later sections, a variety of classes including the Table, Report and Ask classes, will be discussed.



2.2.3 File List

The **Files** tab is a listing of all the classes (.java files) with a brief description of the class.

For example, the Acct.java file will contain a description of the accounting class and its related methods.

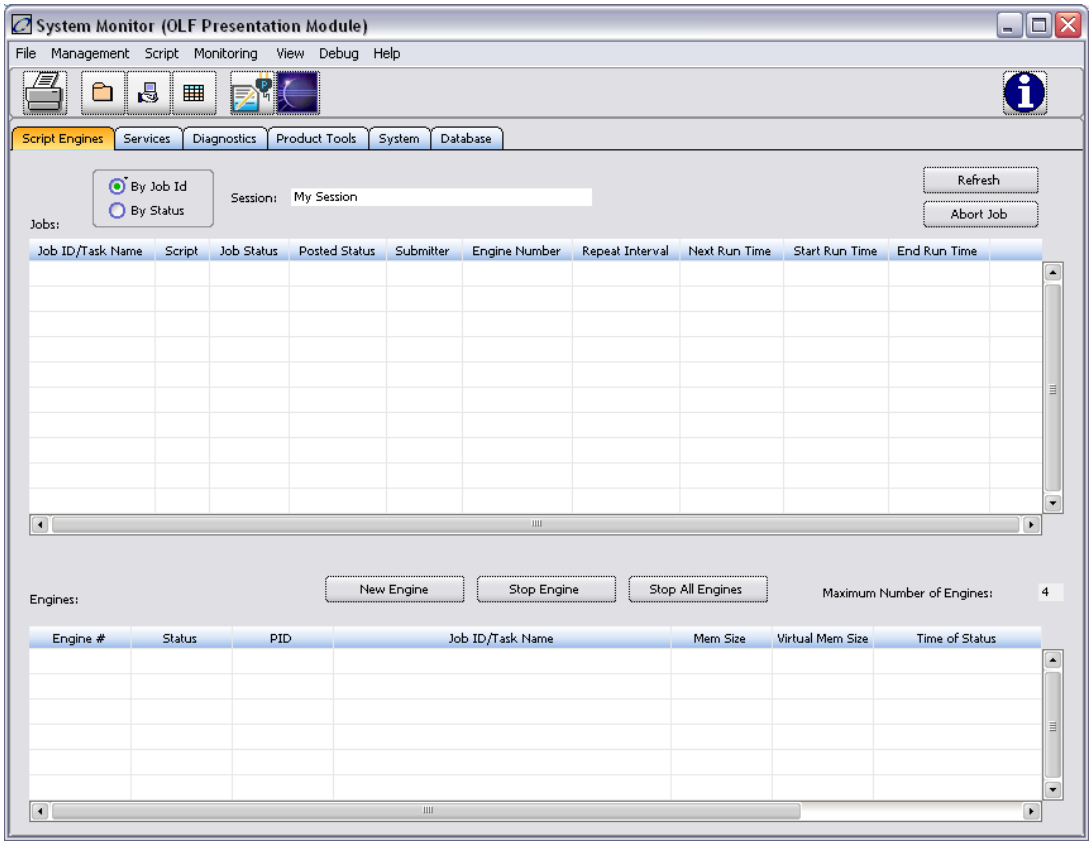


2.3 OpenLink System Monitor

When an OpenJVS plugin is executed, it is run within a script engine. The script engine (SCRENG) is an independent process that can execute OpenJVS plugins. The execution of running jobs (plugins) can be monitored within the System Monitor. An icon is also provided to open Eclipse.

The System Monitor also provides up-to-date status information on the underlying services, engines & technical infrastructure. Some of the uses of this tool as it relates to OpenJVS are:

1. Monitor plugin (Job) status.
2. Diagnose running engines/services.
3. Start/Stop script engines.

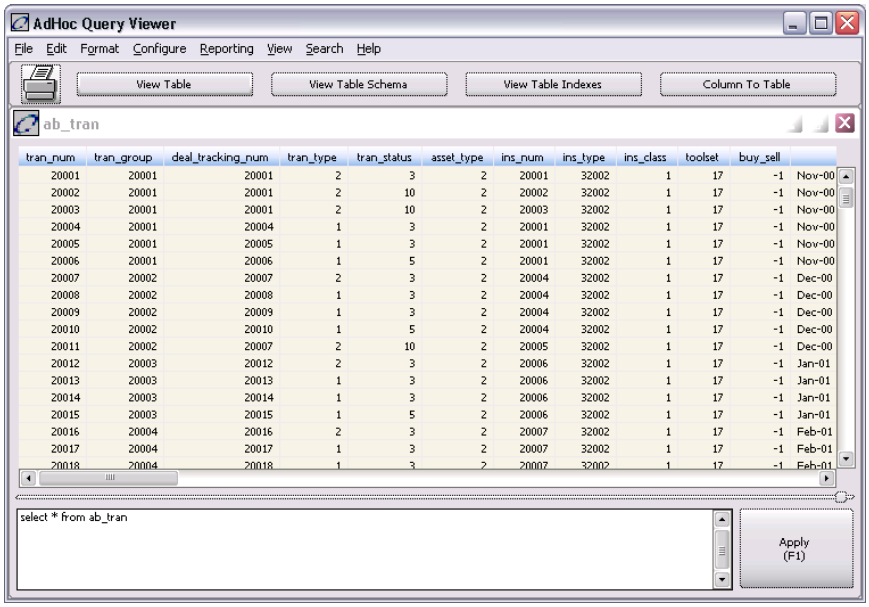


2.4 OpenLink AdHoc Query Viewer

The AdHoc Query Viewer allows system data or user data stored in database tables to be viewed (read-only). A table can be selected from a pick list or a SQL statement can be entered in the bottom panel. Once the criteria is selected or entered, the data can be viewed. The information available in the AdHoc Query Viewer is often needed to retrieve information from the database referenced in a plugin.

The AdHoc Query Viewer allows the user to:

- View database tables
- View the schemas of database tables
- View table indexes
- Identify other tables by selecting a column from the currently displayed table. A pick list will be returned that displays all tables that have that column.



2.5 OpenLink Olisten Console

The Olisten Console typically appears when the user first logs into the OpenLink System. If it is not present at log in, use **Ctrl + F3** to open the window. It is used to:

1. View information and error messages for a local or remote OpenLink System session.
2. Trace application execution for diagnostic purposes (when used in combination with **Ctrl + F8**, enabling various levels of debugging in the application).
3. Gain insights for plugin optimization (e.g., monitor roundtrips to the database).
4. View database calls when opening screens within the OpenLink System.

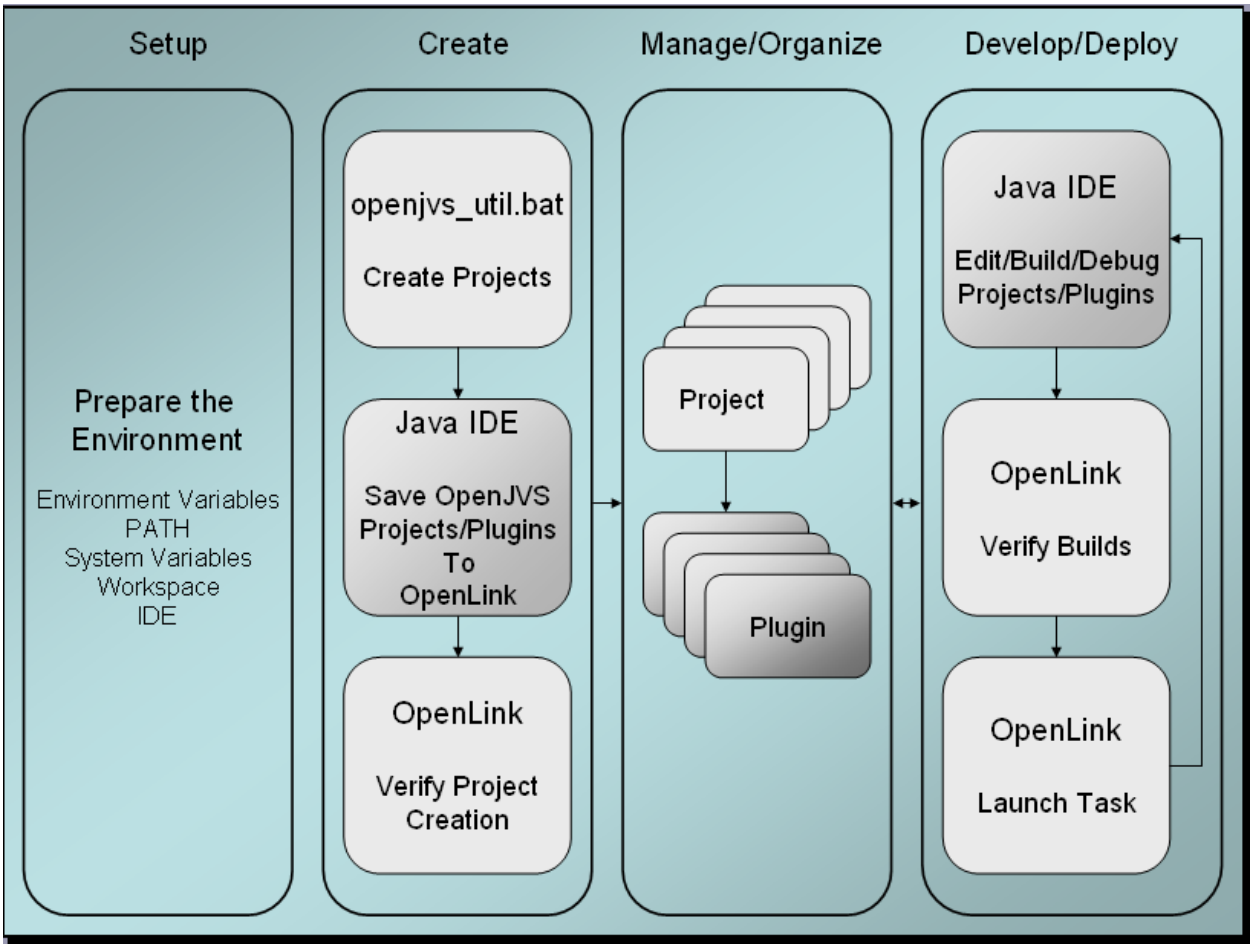
The Olisten console is a useful tool within OpenJVS. Print method return values, tables and error messages are written out to the Olisten console.

3.0 PROGRAMMING CYCLE

The following table and diagram describe the OpenJVS Program Development Cycle.

Setup	Setup activities are required to prepare the environment for development. In addition to environment and OpenLink System variables that must be set, the IDE of choice will require proper setup. Always be sure that setup is checked and verified when upgrading OpenLink.
Create	OpenJVS Projects are created in OpenLink and the Java IDE via a batch file. The project created will have source code and the related files required for building a Java program.
Manage	Create as many OpenJVS projects as required. Each project can sustain one or more plugins.
Develop/Deploy	Much like any other programming cycle (edit-compile-test), plugins are developed and deployed using the JavaIDE and OpenLink.

Edit	Use the Java IDE to edit and develop plugins.
Build	Use the Java IDE to build projects/plugins.
Test	Test as an OpenLink task and debug in the Java IDE
Deploy	Run the plugin using the **Load Task** button or **Script→Load Task** from the menu bar on one of OpenLink's toolsets.



4.0 OpenJVS ENVIRONMENT

This section summarizes the base requirements for setting up the OpenJVS development environment.

4.1 Overview - OpenJVS Software and Configuration Requirements

The following are required for OpenJVS development and are covered in detail in this section:

OpenLink	V82R1_03272008 or above.
Java 1.5.0_15	Bundled with OpenLink V90R1
Ant 1.6.5	Bundled with OpenLink.
A separate Java IDE	Eclipse 3.1.2 and up or NetBeans 5.5.1 and up is recommended for OpenJVS script development. A Java IDE is not bundled with OpenLink and must be downloaded and installed separately from OpenLink.
Settings	Work Directory, Environment Variables, Runtime Variables, PATH and System Variables.

4.2 OpenLink

Required version: V90R1 or above.

4.2.1 Deployed Configuration Items

The following configuration item is created by OpenLink and resides in the OpenLink runtime directory.

openjvs_util.bat	This batch file can be used for creating, deleting, importing, exporting, and listing OpenJVS projects without the need for a Java IDE or compiler. From the OpenLink runtime directory, type openjvs_util to get detailed help.

The following configuration items are created by OpenLink and reside in the OpenLink runtime directory tree under \otk\openjvs\lib.

activation.jar and mail.jar	These provide email capability from within an OpenJVS plugin.
olf_openjvs.jar and olf_openjvs_enums.jar	These contain the OpenJVS API, which serves as the platform for developing custom OpenJVS plugins.
com.olf.openjvs.eclipse.integration_1.0.0.jar	This is the OpenLink Eclipse plugin.
olf_openjvs_ant.jar	This contains the code run by Ant to perform tasks. All the tasks attach to a locally running OpenLink session.
olf_openjvs_build.xml and olf_openjvs_build_global.xml.	These provide the XML directives used by Ant to run the available tasks.

sample_script.java_template	A sample "Hello World" plugin that provides a starting point for new OpenJVS projects.
eclipse_template\.classpath and eclipse_template\.project	These are Eclipse project files that are copied to an Eclipse project directory upon project creation.

4.3 Java

The JRE and a minimal JDK (bin and lib directories only) are bundled with OpenLink and are installed in a subdirectory of the OpenLink runtime directory.

If a full JDK is desired, it must be installed separately. To have OpenLink compile with that JDK the system variable, AB_JRE_HOME, must also be set. See the 'Runtime Variables' section.

4.4 Ant

Ant is bundled with OpenLink and is installed in a subdirectory of the OpenLink runtime directory.

Apache Ant is an open source 3rd party Java build tool. It serves as the integration platform for launching OpenLink tasks to support the needs of the OpenJVS environment.

Ant requires the environment variable JAVA_HOME to be set to a JDK directory.

4.5 Workspace (directory)

The Workspace is a directory where projects will be saved. The directory can be created on a local or network drive.

4.6 Environment Variables

The following environment variable is utilized for release and development of build sets:

4.6.1 JAVA_HOME

This variable used by the Java class loader. If it is not set, the class loader will issue a warning message when running tasks from openjvs_util.bat. See the 'Error Messages' section for a description of this message. Note the following:

- nn = 12 or 15 depending on the version of OpenLink.
- If the JDK bundled with OpenLink is used, this variable can be set to:

  cut\bin\olf_dependencies\java\jdk1.5.0_nn

  Example: D:\olf\bin\olf_dependencies\java\jdk1.5.0_nn
- If a separate JDK is installed, set this variable to the JDK's root directory.

  Example: C:\Program Files\Java\JDK1.5.0_nn
- The value of JAVA_HOME must not be surrounded by double quotes, even if it contains spaces. This is not an OpenLink system variable.

4.7 Runtime Variables

OpenLink bundled JDK (minimal JDK installation).

nn = 12 or 15 depending on the version of OpenLink.

AB_JRE_HOME defaults to olf_dependencies\java\jdk1.5.0_nn\jre

```
.\olf_dependencies\java\jdk1.5.0_nn      <== JAVA_HOME
                          +-- bin
                          +-- jre        <== AB_JRE_HOME
                               +-- bin
                          +-- lib
```

Separate JDK Installation

```
.\jdk1.5.0_nn                <== JAVA_HOME
   +-- bin
   +-- jre                   <== AB_JRE_HOME
        +-- bin
   +-- lib
```

4.8 PATH – Release and Development Build Sets

The Java IDE executable location must be in your path. In a common extract this is:

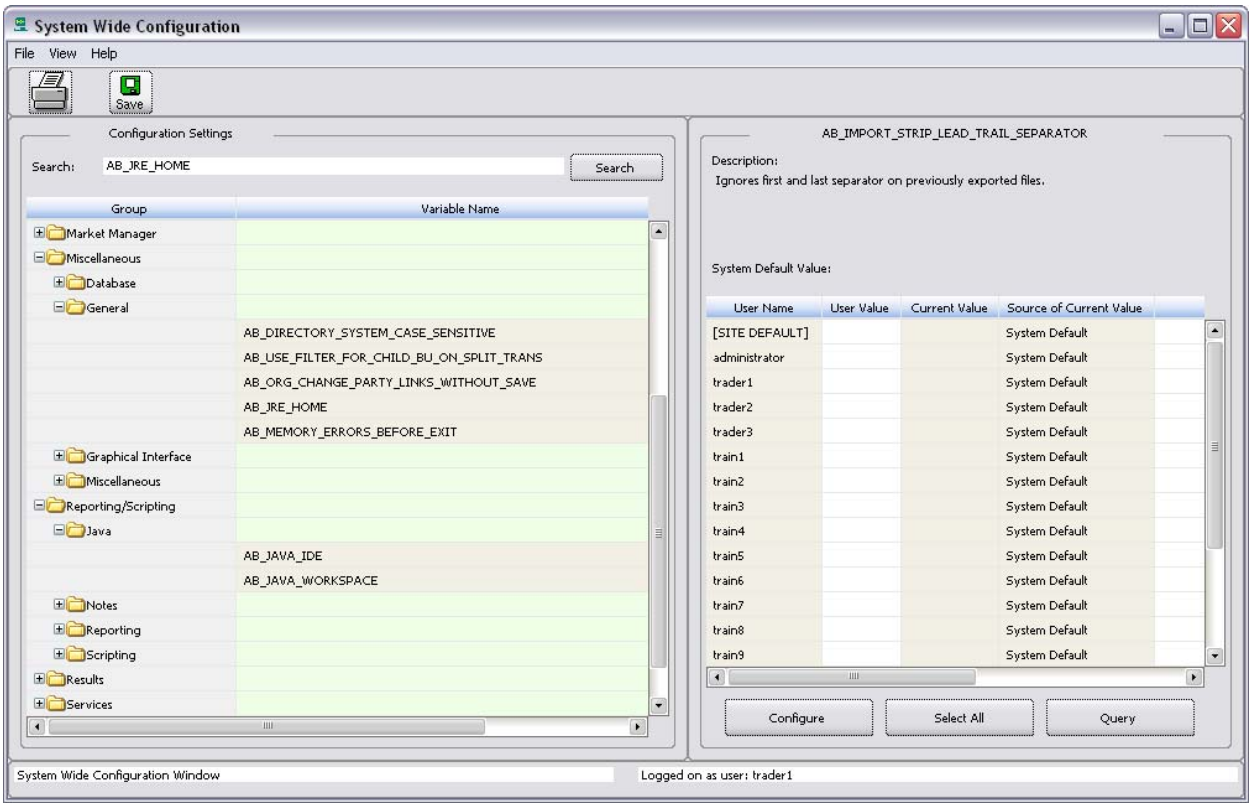
Eclipse	X:\Program Files\Eclipse
NetBeans	X:\Program Files\NetBeans n.n.n\bin

4.9 System Variables

The following OpenLink System variables are utilized:

AB_JAVA_IDE	for development build sets.
AB_JAVA_WORKSPACE	for development build sets.
AB_JRE_HOME	for both release and development build sets.

System Variables can be viewed and/or set via **Admin Manager -> System Wide Configuration**.



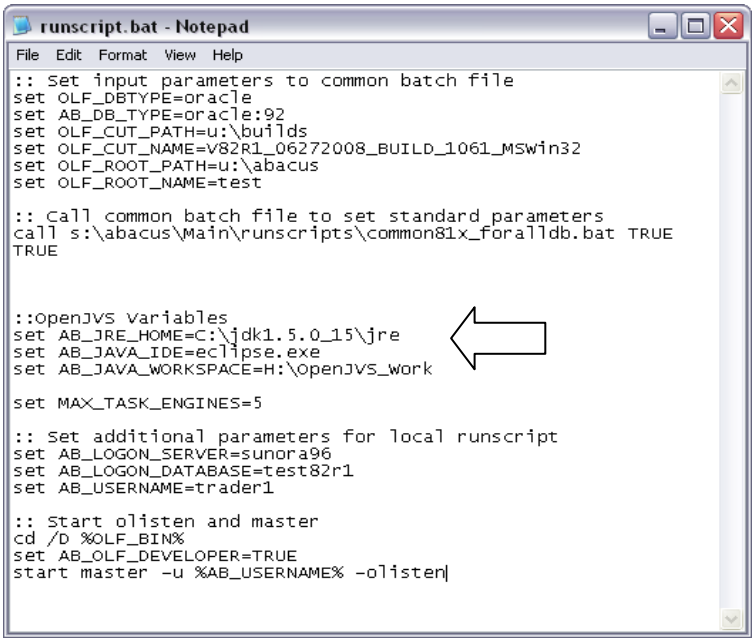
They can also be set within a runscript batch file for OpenLink as shown below.

4.9.1.1 AB_JAVA_IDE

This variable is used to configure the command line statement to launch a Java IDE application.

If the IDE's executable is in your path, type the executable name; eclipse.exe for example. If the IDE's executable is not in your path, type the absolute path and executable name.

(Note: the NetBeans executable, nb.exe, is not recognized by OpenLink. netbeans.exe must be used.)



There are also three keywords that can be used in the variable's value. These keywords are replaced with the following values before the command is executed:

Keywords	Replaced With
openjvs.project.name	The name of the OpenJVS project to which the script belongs.
openjvs.script.file	The absolute path and filename to where the script will be exported. The AB_JAVA_WORKSPACE variable provides the path and the script provides the filename.
openjvs.project.basedir	The AB_JAVA_WORKSPACE variable value.

For example,

AB_JAVA_IDE is set to: notepad.exe openjvs.script.file
AB_JAVA_WORKSPACE is set to: D:\OpenJVS
A script named "SampleScript" is being edited
The resulting command line executed would be:
notepad.exe D:\OpenJVS\SampleScript.java

The integration with Eclipse|NetBeans provided by OpenLink does not require these keywords to be present. Simply setting AB_JAVA_IDE to eclipse.exe/netbeans.exe is sufficient for OpenLink to process the request. Note the following:

- If AB_JAVA_IDE is not configured, scripts can only be accessed in read-only mode in the *OpenLink Script Editor* window.

4.9.1.2 AB_JAVA_WORKSPACE

This variable represents the base absolute path to where all OpenJVS projects and plugins will be exported. This path does not need to exist; OpenLink will create it for you, but you do need to have write permission to its starting location. Note the following:

- This OpenLink variable is used during plugin editing tasks.
- If this OpenLink variable is not set, your 'home' directory will be used. In Windows, this is generally the USERPROFILE environment variable. Your particular installation may vary.

4.9.1.3 AB_JRE_HOME

This variable is blank by default. In this state, OpenLink will use the bundled JDK and JRE. If an installed, separate JDK is desired, AB_JRE_HOME must be set to the JDK's JRE directory for OpenLink to utilize it. Note the following:

- nn = 12 or 15 depending on the version of OpenLink.

  Example: C:\Program Files\Java\jdk1.5.0_nn\jre
- This variable is located in OpenLink's Admin Manager / System Wide Configuration.

5.0 OpenJVS IDE INTEGRATION

The use of a Java IDE is optional, but recommended to fully utilize the OpenJVS script development environment. OpenLink provides integration with Eclipse|NetBeans through custom XML files and the openjvs_util.bat file. When creating an OpenJVS project with this batch file using either –c or –C option, the OpenJVS infrastructure will set up a properly formed Eclipse|NetBeans project in the Java workspace. This project will contain project.xml and build.xml files that provide OpenJVS menu options for managing OpenJVS projects.

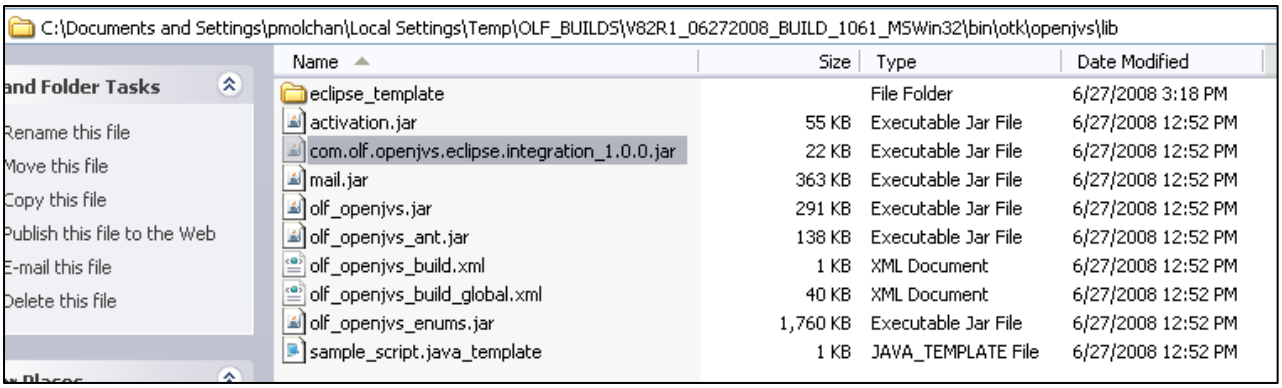
5.1 Eclipse

Eclipse can be downloaded from http://www.eclipse.org. It is a zip file and is not installed but extracted. A common extract location on a Windows-based computer is C:\Program Files.

The Eclipse executable location must be in your path. In a common extract this is: C:\Program Files\Eclipse.

5.1.1 Eclipse Configuration

An Eclipse plug in is provided that allows for OpenJVS development in Eclipse. Copy the following:
OpenLink_installation_directory\bin\otk\openjvs\lib\com.olf.openjvs.eclipse.integration_1.0.0.jar
to:
eclipse_installation_directory\plugins



5.1.2 Adding OpenJVS Java Docs

This is automatically handled by the OpenLink Eclipse plugin.

5.1.3 Refreshing OpenLink's Eclipse Plugin

When you receive a new OpenLink System cut with an updated plugin, you must copy the new plugin to the plugins directory as stated in the Configuring step.

Additionally, the first time you launch Eclipse, a command line option, -clean, must be specified to refresh Eclipse's plugin cache.

5.2 NetBeans

NetBeans can be downloaded and installed from http://www.netbeans.org.

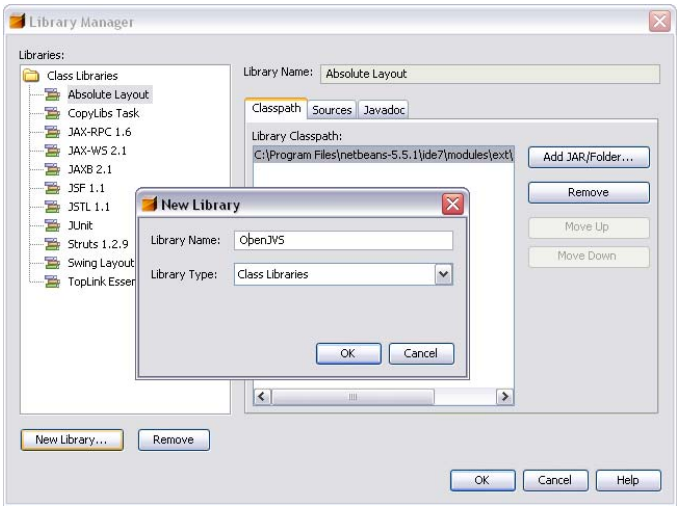
The NetBeans executable location must be in your path. In a default installation this is: C:\Program Files\netbeans n.n\bin, where n.n is the NetBeans version.

5.2.1 NetBeans Configuration

1. Open netbeans n.n/etc/netbeans.conf in a text editor.
2. Set 'netbeans_default_options' to: (include the quotes)
   "-J-Xms64m -J-Xmx256m -J-XX:PermSize=32m -J-XX:MaxPermSize=160m -J-Xverify:none -J-Dapple.laf.useScreenMenuBar=true"
3. Set 'netbeans_jdkhome' to the same value as JAVA_HOME (include the quotes)
   Example: "D:\olf\bin\olf_dependencies\java\jdk1.5.0_nn"
4. Save and close the file.

5.2.2 Adding OpenJVS Javadocs

1. From the NetBeans menu bar, select **Tools→Library Manager**.
2. From the Library Manager dialog, click the **New Library…** button in the lower left corner.
3. Type **OpenJVS** for the Library Name and click **OK**.
4. Under the Classpath tab, add the following four jar files located under <OpenLink System runtime directory>\otk\openjvs\lib:activation.jar, mail.jar, olf_openjvs.jar, and olf_openjvs_enums.jar.



5. Under the Javadoc tab, add the following jar file located under < OpenLink System runtime directory>\otk\openjvs\docs:olf_openjvs_docs.jar
6. Click **OK**.

   Important note:

   In NetBeans 6.0 and above, there is a bug that prevents a method's argument names from displaying properly in the JavaDocs. The descriptive argument names are replaced with generics: arg0, arg1, arg2 ...

   Example: table.addCol(String col_name, String title, int width, int prec, int base) displays as: table.addCol(String arg0, String arg1, int arg2, int arg3, int arg4)The link to the bug: http://www.netbeans.org/issues/show_bug.cgi?id=85329

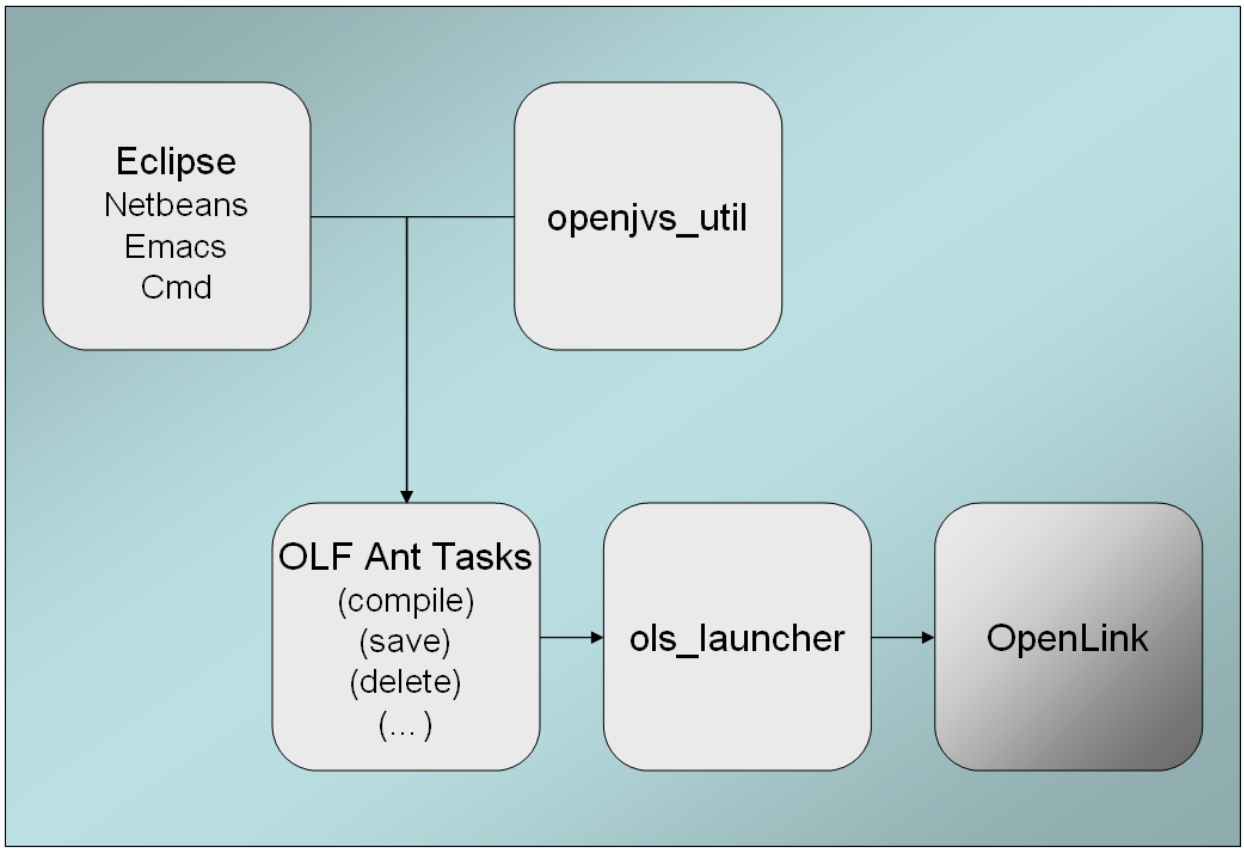
6.0 OpenJVS - HOW THE PIECES FIT TOGETHER

The starting point for OpenJVS development is running an Apache Ant task. An Apache Ant task can be run from the command line using openjvs_util.bat. In the Eclipse|NetBeans IDE environment, the project must have been created using openjvs_util.bat for the OpenJVS menu options to be available.

The menu options are provided by the project.xml file. This file calls an Ant task in the build.xml file to process the request.

The Ant task may call one or more custom Java tasks that reside in olf_openjvs_ant.jar. The Java classes create temporary task request files.

The Java classes send requests to OpenLink with the names of request files through the AFE service. OpenLink processes the files. If successful, OpenLink returns a succeeded status code to the Ant task. If not successful, OpenLink returns a failed status code to the Ant task along with the error message. The temporary request files are then deleted.



7.0 OpenJVS_UTIL.BAT

When creating an OpenJVS project with this batch file the OpenJVS infrastructure will set up a properly formed Eclipse|NetBeans project in your Java workspace and in OpenLink. This project will contain project.xml and build.xml files that provide OpenJVS menu options for managing your OpenJVS projects.

openjvs_util.bat provides all the available OpenJVS command line functionality. **OpenLink** must be running when openjvs_util.bat is executed.

openjvs_util.bat must be run from the OpenLink runtime directory, unless the –o optional flag along with the absolute path to the OpenLink runtime directory is provided.

Example –o d:\olf\bin.

If the path to the OpenLink runtime directory contains spaces, CD to the short path name in the DOS window to run the openjvs_util.bat command line utility.

The path specified in the –o option cannot contain spaces. If the path does contain spaces, the short path name must be used – see the example below.

Example: C:\Progra~1\olf\bin instead of C:\Program Files\olf\bin.

All successfully processed tasks will return a "**BUILD SUCCESSFUL**" message to the command window.

8.0 WALKTHROUGH: CREATING AN OpenJVS PROJECT AND RUNNING A SAMPLE PLUGIN

This section steps through creating an operational OpenJVS project and running a sample plugin. Note that the steps listed in the OpenJVS Development Steps below will be detailed.

OpenJVS Development Steps

1. Prepare for Project Development (one time).
2. Use openjvs_util.bat to create your OpenJVS project shell.
3. Use Eclipse|NetBeans IDE to develop plugin(s) in the project.
4. Verify project creation in OpenLink.
5. Use Eclipse|NetBeans to Build the project.
6. Verify project build in OpenLink.
7. Use OpenLink to Run the plugin.
8. Use Eclipse|NetBeans to Debug the plugin.

8.1 Prepare for Project Development

Prior to starting actual project development, the following preparations covered in previous sections of this guide need to be completed.

Initial Startup Item	Section Described	Description
Workspace	4.5	Create a directory to serve as a workspace directory. Example X:\OpenJVS_Work
Environment Variable	4.6	Set the environment variable used by the Java class loader: JAVA_HOME
PATH	4.8	Add the IDE bin directory: Eclipse X:\Program Files\Eclipse NetBeans X:\Program Files\NetBeans n.n.n\bin
System Variables	4.9	Set the following OpenLink System variables: AB_JAVA_IDE AB_JAVA_WORKSPACE AB_JRE_HOME
IDE Integration	5.0	Install Eclipse plugin or configure NetBeans as required.
Start OpenLink		OpenLink must be running to execute openjvs_util.bat in the next step.

8.2 Create a Project (command line)

Open a command prompt and set your location to the \olf\bin directory. The command syntax and an example are shown below:

openjvs_util –C <project name> –w <absolute path to workspace> -t < eclipse|netbeans>

Example: openjvs_util –C Exercise –w D:\OpenJVS_Work –t eclipse

This will create a project named Exercise in OpenLink as well as a fully configured sample project under D:\OpenJVS_Work. Only alphas and underscores are permitted for project names.

–w	<absolute path to workspace> flag will create an OpenLink compatible project under the path along with a sample "Hello World" OpenJVS plugin. This project is Eclipse|NetBeans compatible and contains the necessary OpenLink integration files.
–C	(uppercase will create the local project and a project entry in OpenLink. OpenJVS project creation can also be performed with the –c (lower case) flag. This operates identically to the–C (upper case) flag, but without creating the OpenLink project entry. The local project will still be OpenLink and Eclipse|NetBeans compatible and contain the OpenLink integration files.
–t	Specifies the Java IDE as Eclipse or Netbeans.

Also created in this process are the OpenLink specific files **jproject.classpath** and **<projectname>.out**.

8.2.1 jproject.classpath

jproject.classpath is a text file containing a list of other OpenJVS projects in Endur/Findur on which the project depends. It is located under the OpenJVS project directory. It is a required file for importing an OpenJVS project into Endur/Findur and is automatically created during an export task. Its contents are updated through the 'Update Project References in Endur/Findur' task. ***Do not edit this file manually***.

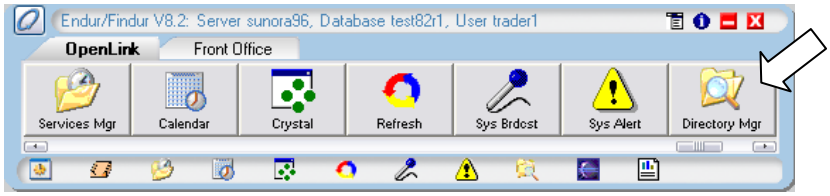
8.2.2 <projectname>.out

<projectname>.out is a text file containing the project's name, source files and other information required for importing an OpenJVS project into Endur/Findur. It is located in the same directory as the project directory and is automatically created during an export task. Its contents are updated through several tasks. ***Do not edit this file manually***.

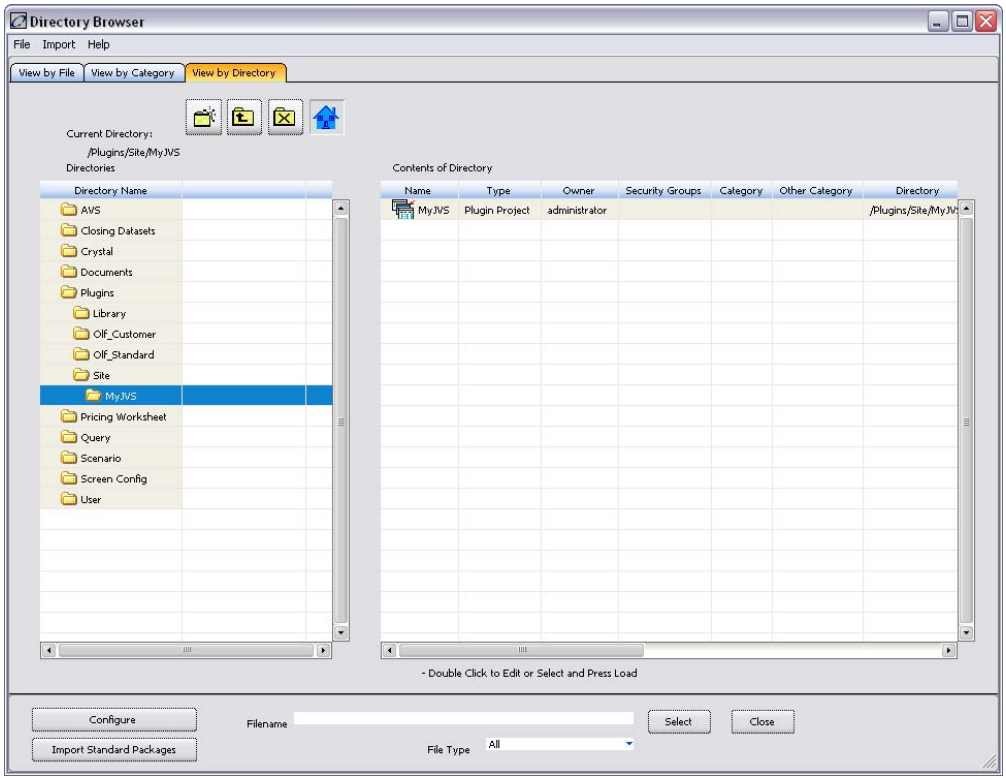
8.3 Verify Project Creation in OpenLink

The project in OpenLink will have been created only if the –C (upper case) was used in the previous step.

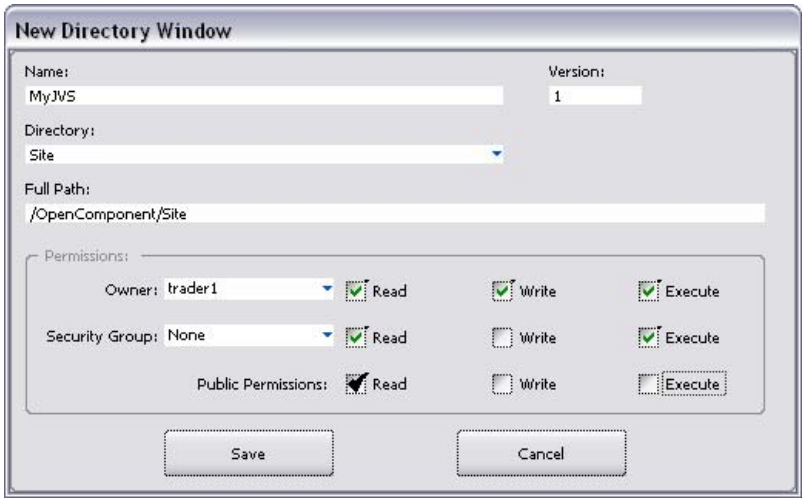
1. From OpenLink's Master Central, select **Directory Manager**.



2. Following the Directory Tree on the left panel, click **Plugins**, **Site**, **Project Name** to see the following screen:



Projects are created under Plugins/Site and have the default permissions shown in the illustration below. Double click the new project to open and review the default settings of the *New Directory* window.



8.4 Build a Project

Building the project from Eclipse or NetBeans will complete creation of the fully configured project in OpenLink.

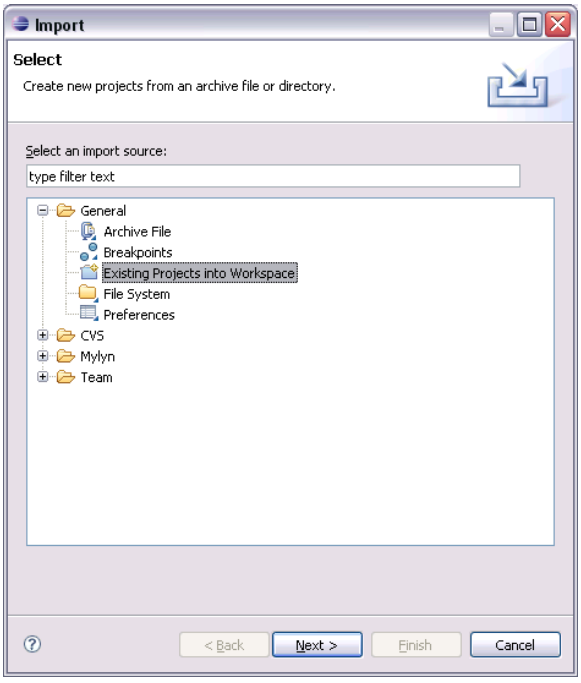
The following sections walk through the steps required to build an OpenJVS project with Eclipse and with NetBeans.

8.4.1 Eclipse

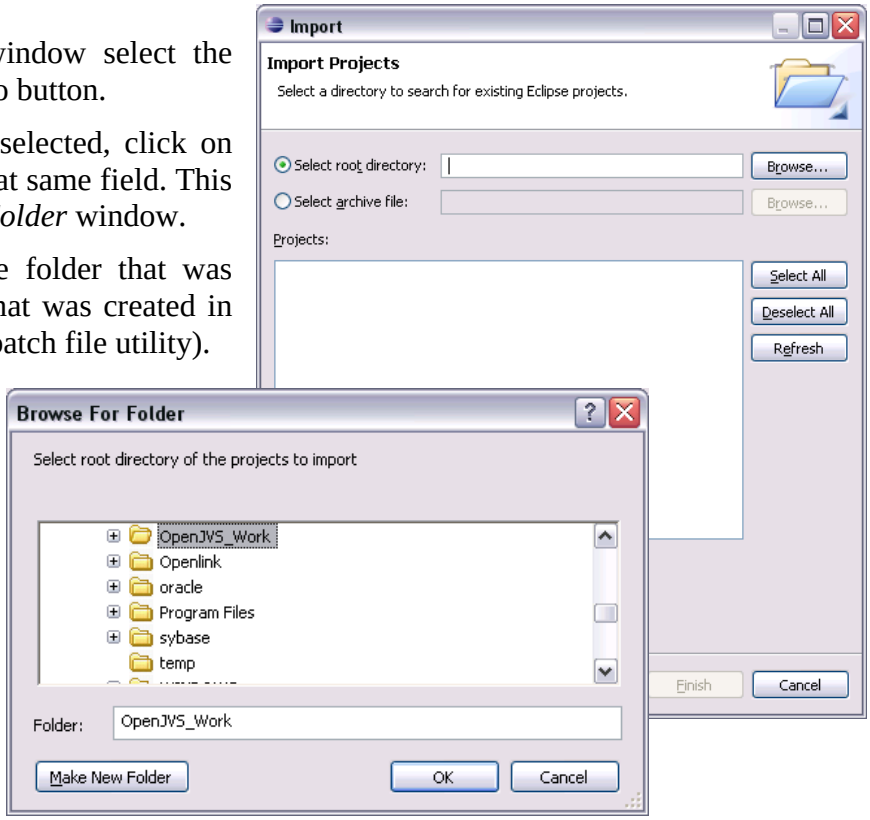
Follow the steps outlined below to build a project using Eclipse.

8.4.1.1 Import the Project into Eclipse

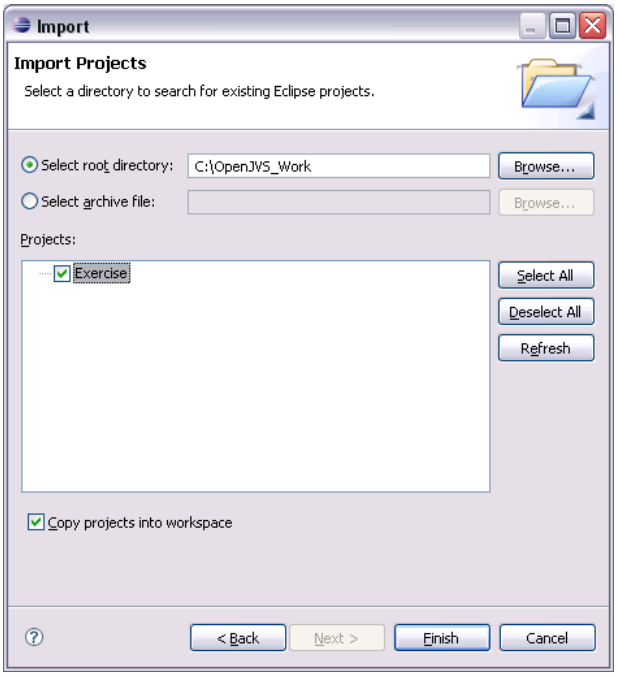
1. If Eclipse has not been configured with the OpenComponents plug in, go back to the *Eclipse Configuration* section and follow the instructions there.
2. Launch Eclipse.
3. From the menu bar in the Eclipse script editor select **File→Import**. The *Import* window will open.
4. From the *Import* window expand the **General** folder and select **Existing Project into Workspace.**
5. Click the **Next** button. The *Import Projects* window +appears.



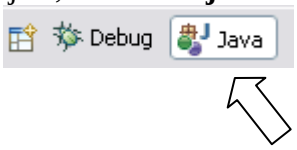
6. In the *Import Projects* window select the **Select root directory** radio button.
7. Once the radio button is selected, click on the **Browse…** button of that same field. This will open the *Browse for Folder* window.
8. In this window select the folder that was used to save the project that was created in the earlier step (using the batch file utility).

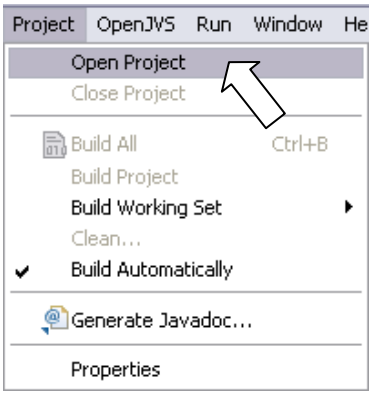


9. When the folder is selected click on the **OK** button
10. In the *Import Projects* screen select the project that was created in the earlier step (Ex: Exercise).
11. Once the project is selected, click on the **Finish** button. The *Import* window closes.

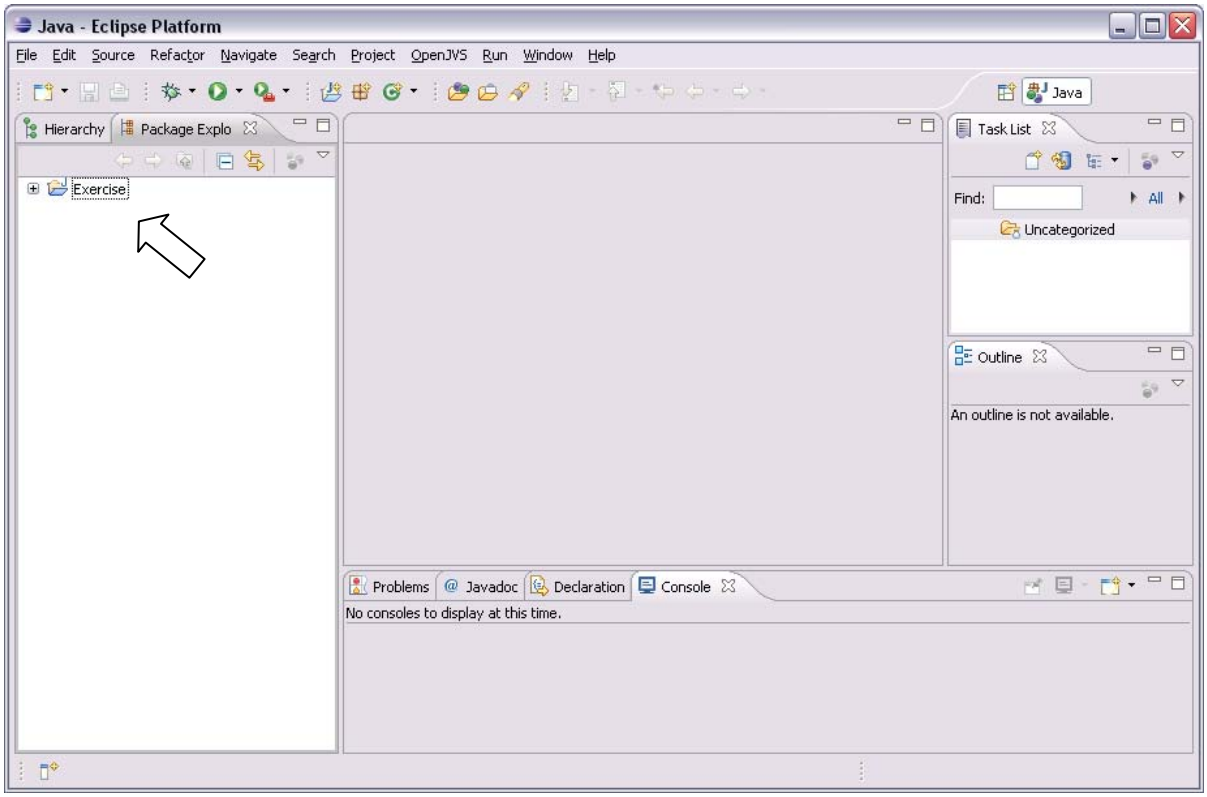


8.4.1.2 Open the Project with Eclipse

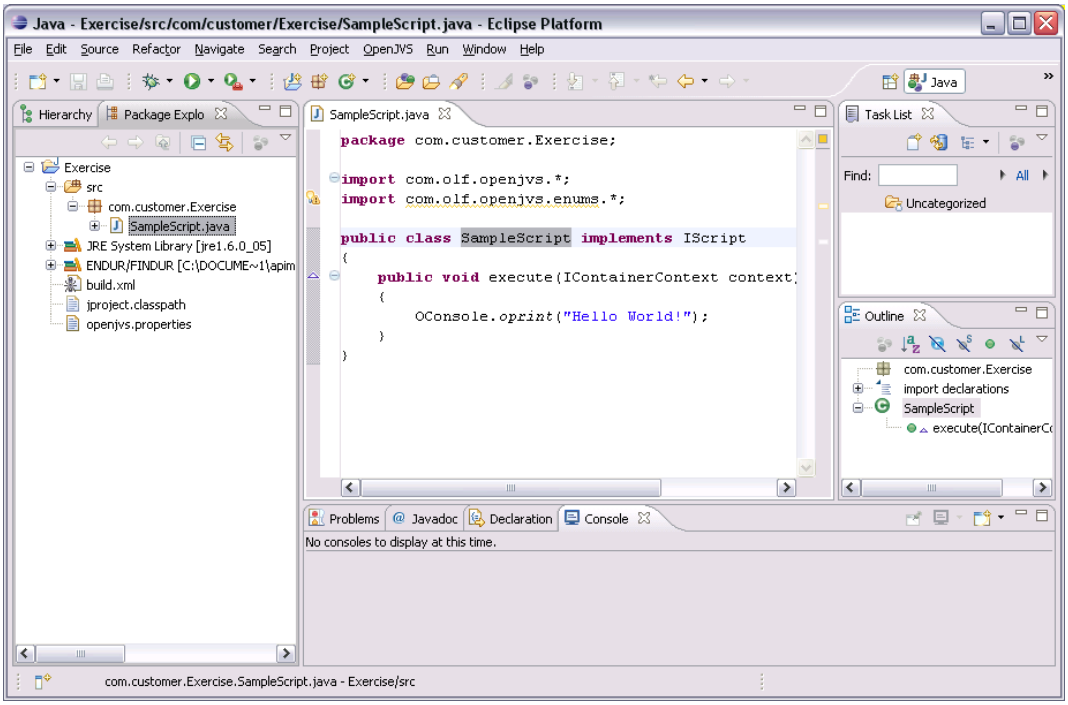
12. To open the newly imported project, select **Project→Open-Project**. (Note: be sure you are viewing in the Java Perspective. 



13. The project will display in the *Package Explorer* panel of the Eclipse platform.



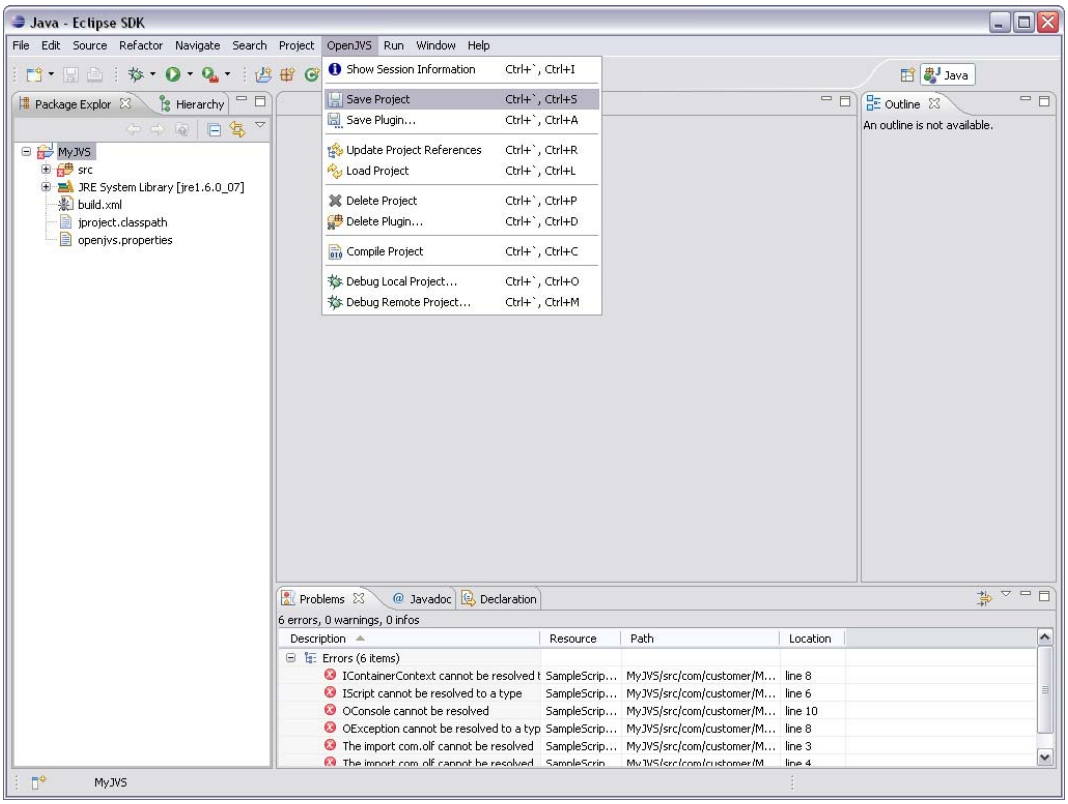
14. Once the project is opened, expand it and view the sample script created along with the project. In this sample script, "Hello World" will be sent (printed) to the Olisten Console.



8.4.1.3 Build the Project with Eclipse

15. Click **OpenJVS→Save Project**.

16. The output panel at the bottom of the IDE will provide a listing of the steps being executed and any success or failure messages.

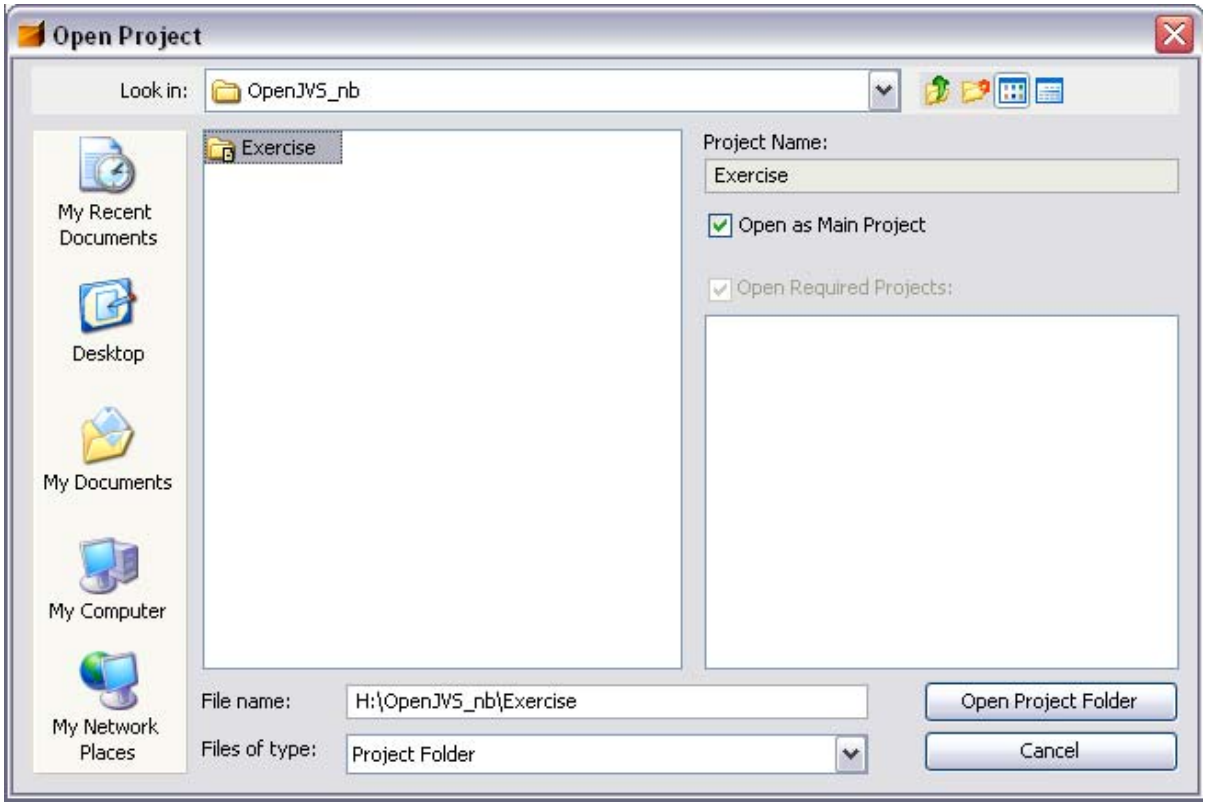


8.4.2 NetBeans

Follow the steps outlined below to build a project using NetBeans.

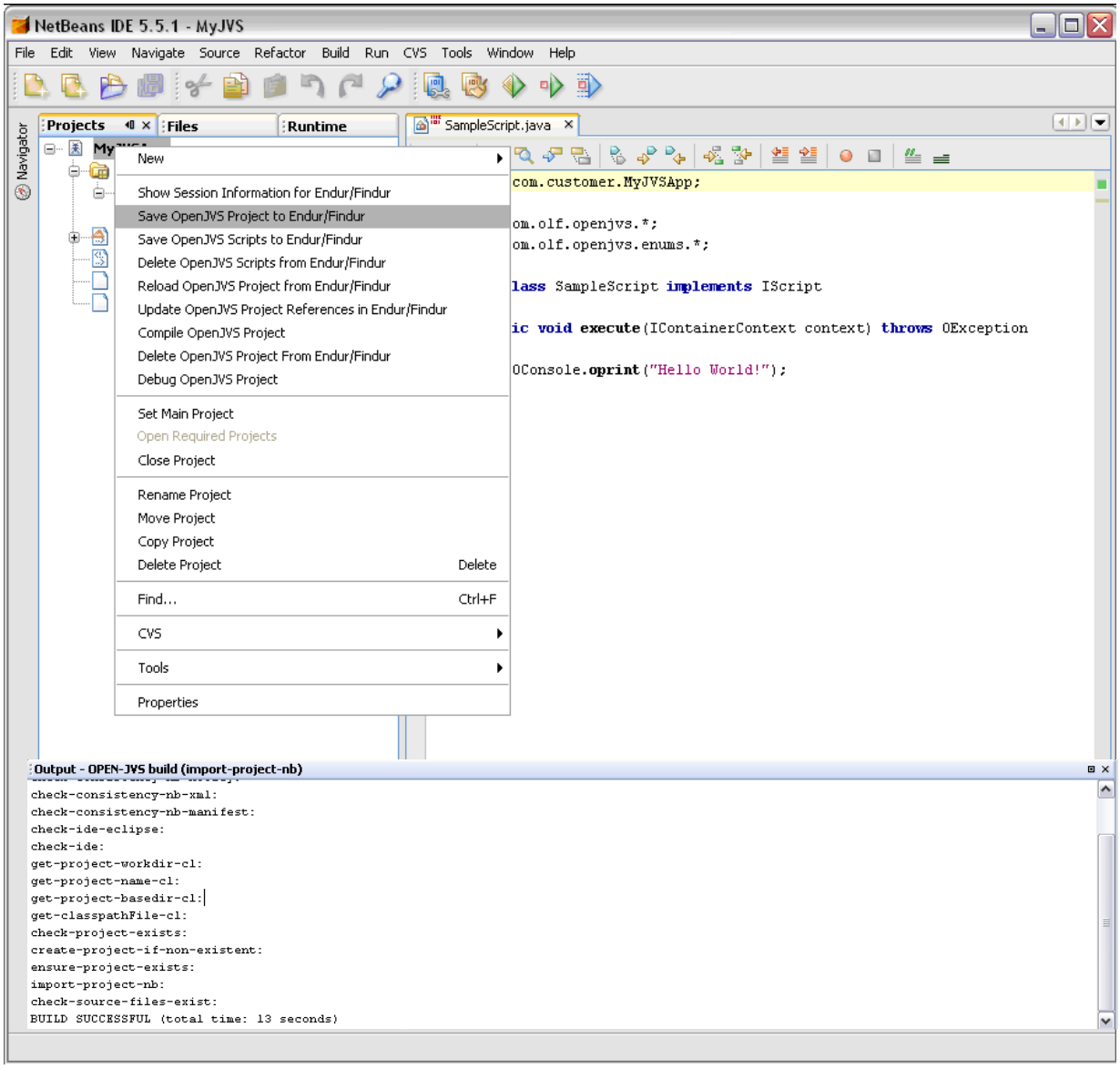
8.4.2.1 Open the Project with NetBeans

1. If NetBeans has not been configured, go back to the *NetBeans* section and follow the instructions there.
2. Start NetBeans.
3. Click **File→Open Project**.
4. Select the project folder, check **Open as Main Project** and click the **Open Project Folder** button.



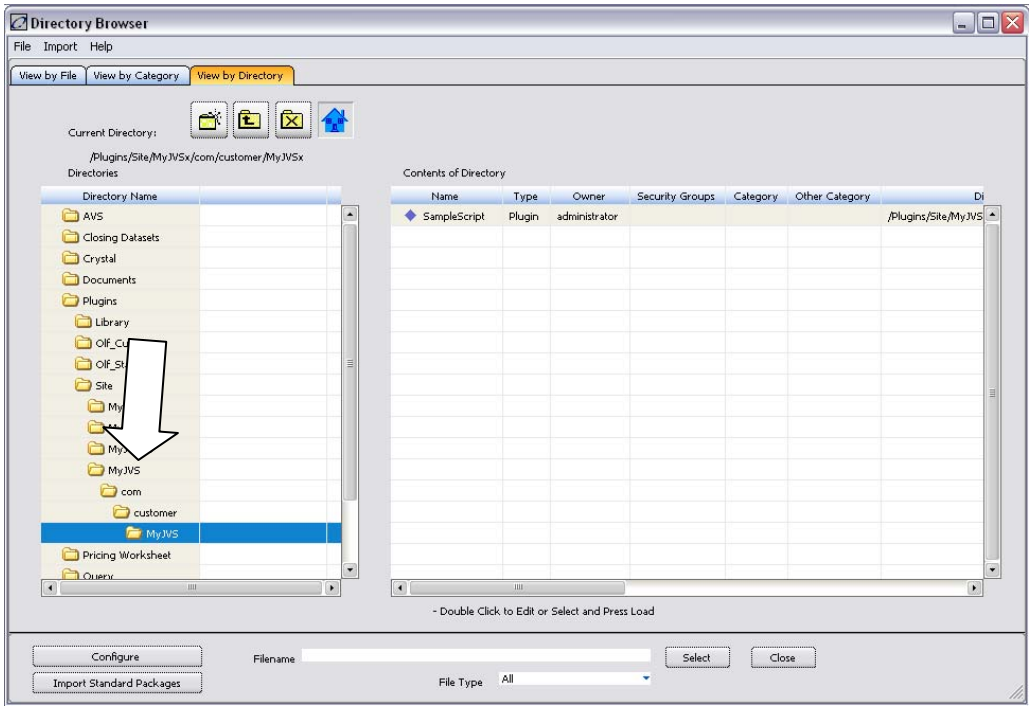
8.4.2.2 Build the Project with NetBeans

1. From the **Projects Explorer** tab, right-click on the project entry to get the OpenLink integration pop-up menu. Select **Save OpenJVS Project to Endur/Findur**.
2. The output panel at the bottom of the IDE will provide a listing of the steps being executed and any success or failure messages.



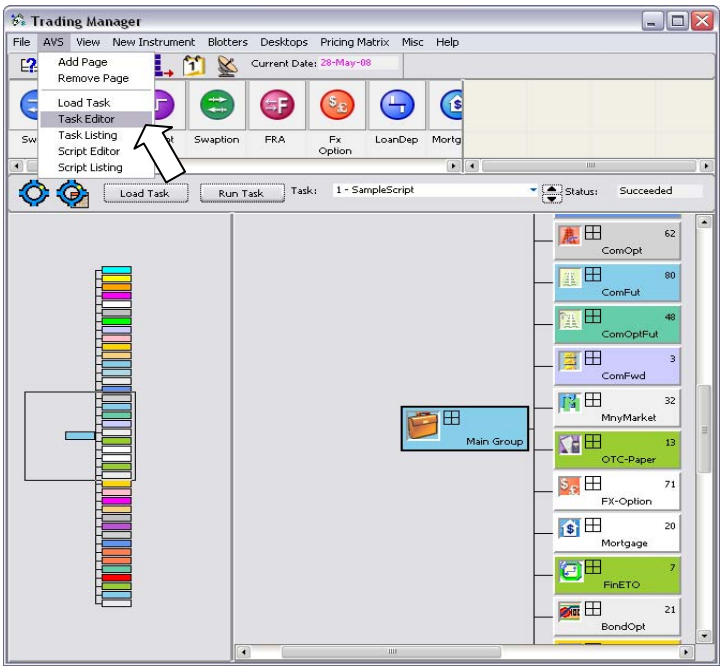
8.5 Verify Project Build in OpenLink

1. Go to the OpenLink Directory Manager.
2. The **Plugins/Site/MyJVS/com/customer/Project Name** subdirectory tree will have been created. The plugin, SampleScript will be listed in the right-hand panel after clicking on the lowest level directory.

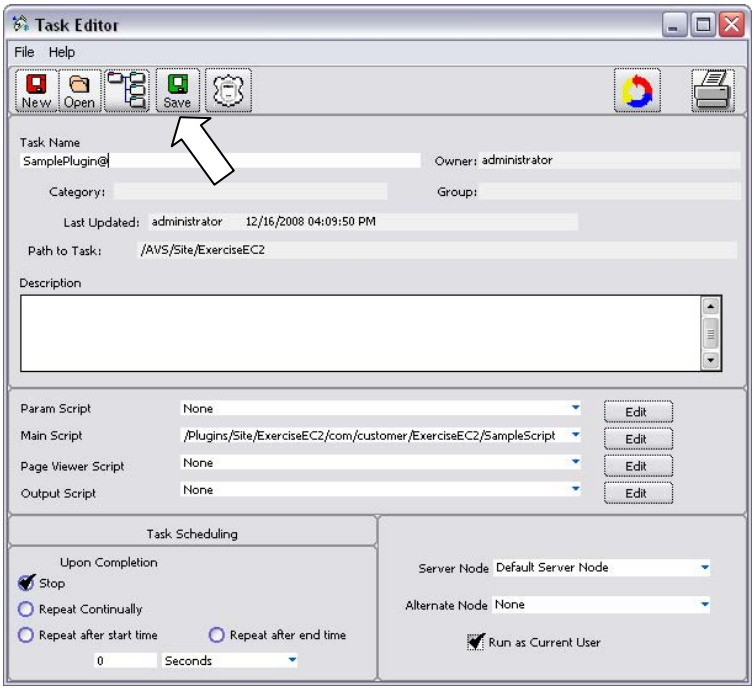


8.6 Run a Plugin in OpenLink

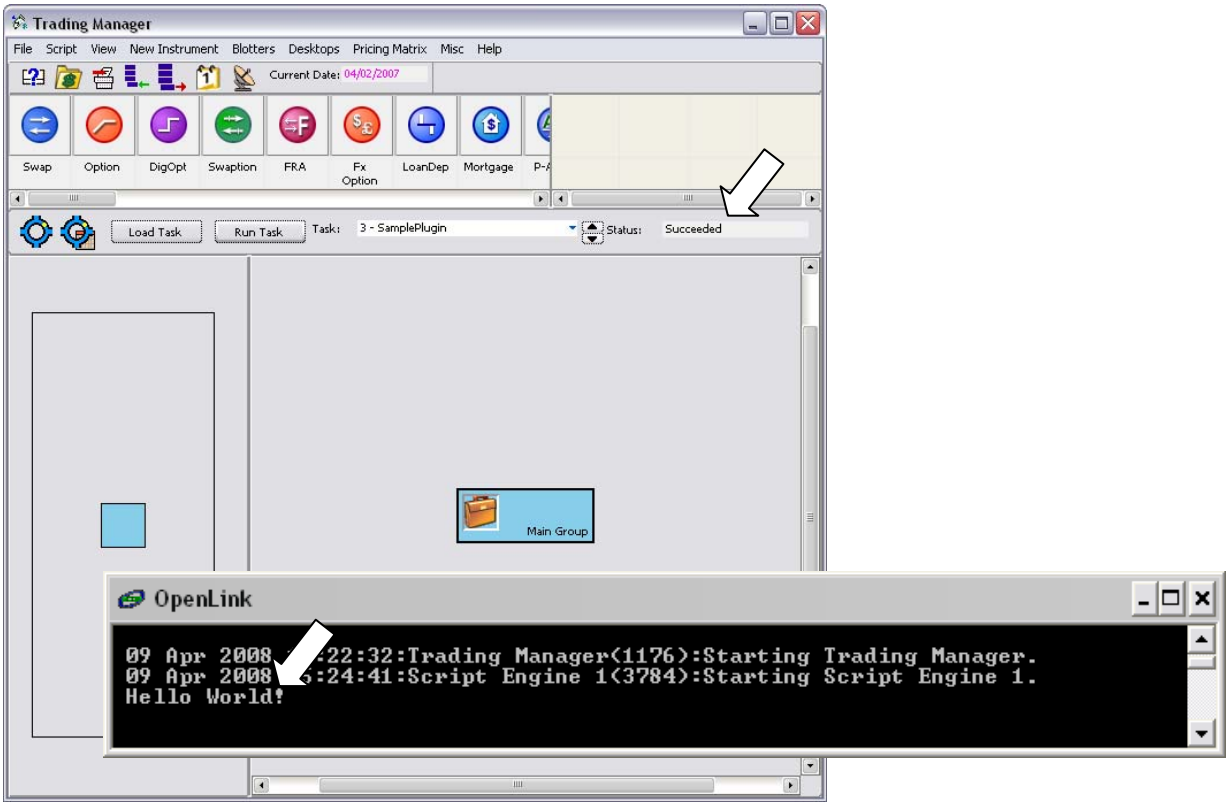
1. Select **Script →Task Editor** from the menu bar on one of OpenLink's toolsets. Shown here is the Trading Manager.
2. Click the down arrow in the **Main Script** list box and select the **Exercise** script that was just imported. Name the task. Ex. "SamplePlugin".



3. Click the **Save** button at the top of the window and close the window when the save is completed.



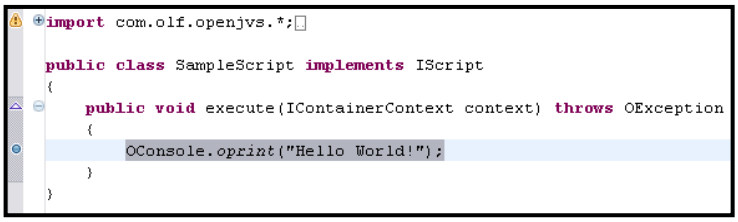
4. OpenLink will automatically load SampleScript as the current task.
5. Click the **Run Task** button. Two events will occur – the Status will progress "**Blocked – Running - Succeeded**" and the *OListen* window will contain the **"Hello World"** message.



8.7 Debug a Plugin (Eclipse or NetBeans IDE Only)

Plugins can only be debugged after they have been built in OpenLink.

8.7.1 Set a Breakpoint in the Plugin



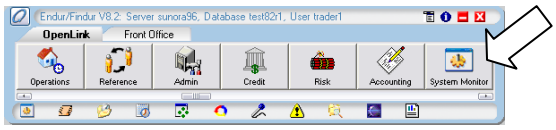
1. In the Eclipse|NetBeans project, set a breakpoint at the OConsole.oprint("…") line in SampleScript.java.
2. **Save** the Project.

Eclipse	**OpenJVS→Save Project**
NetBeans	**Save OpenJVS Project to Endur/Findur (**from the project's pop-up menu.).

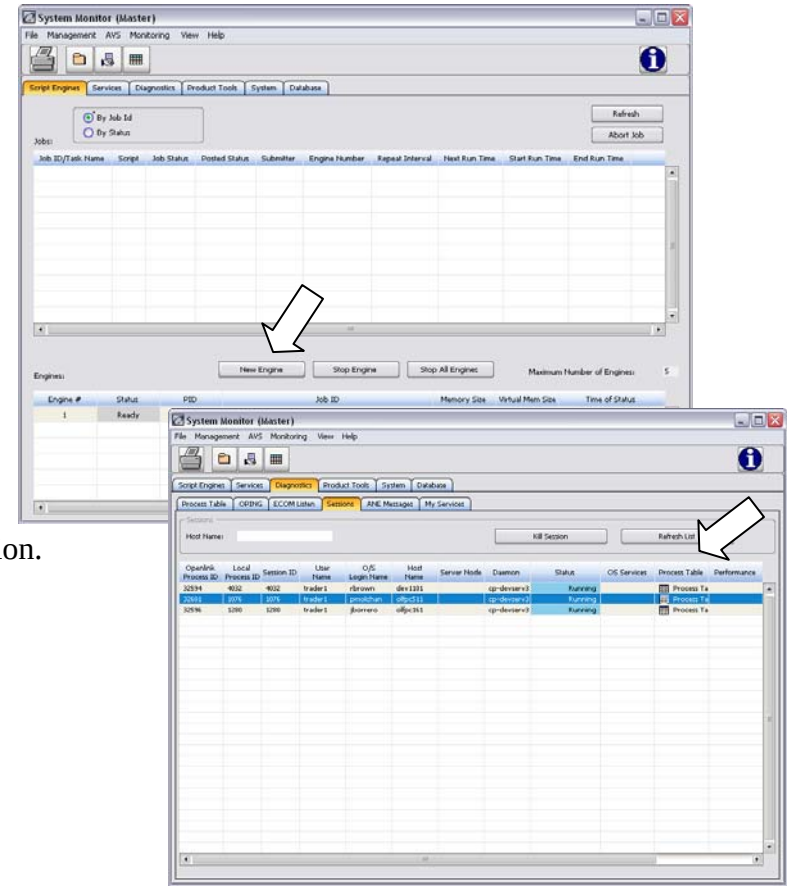
8.7.2 Start and Initialize an OpenLink Script Engine

Follow the steps below to start an OpenLink script engine and initialize a JVM.

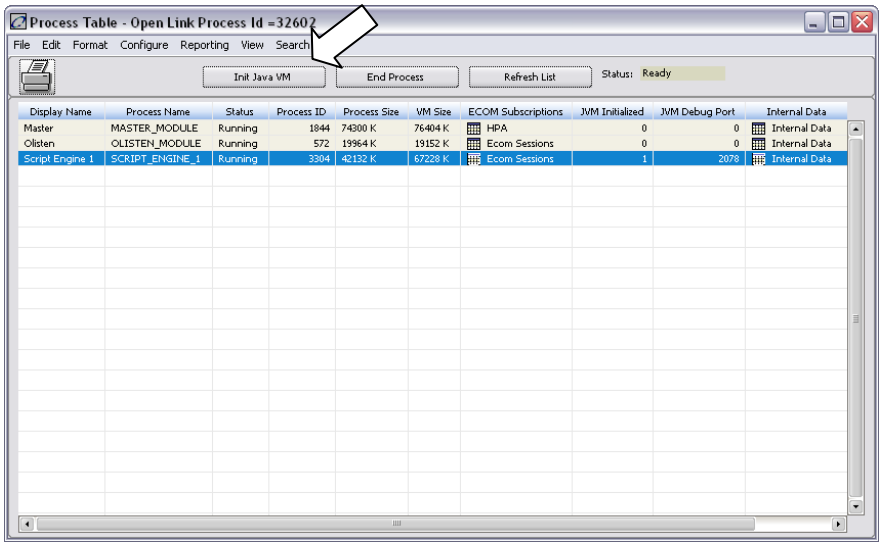
1. From OpenLink's Master Central, select the **System Monitor**.



2. Click the **New Engine** button on the **Script Engines** tab.
3. When the screen engine displays "Ready", select the **Diagnostics** tab and then the **Sessions** tab of the *System Monitor* window. The session in which the new engine was created will have the user name that was used to log into the OpenLink System and the login name that was used to log into the operating system of the machine.
4. Double-click on the **Process Table** field of the applicable session.



5. Select the new script engine and click the **Init Java JVM** button. Note the **JVM Debug Port** number that is returned.

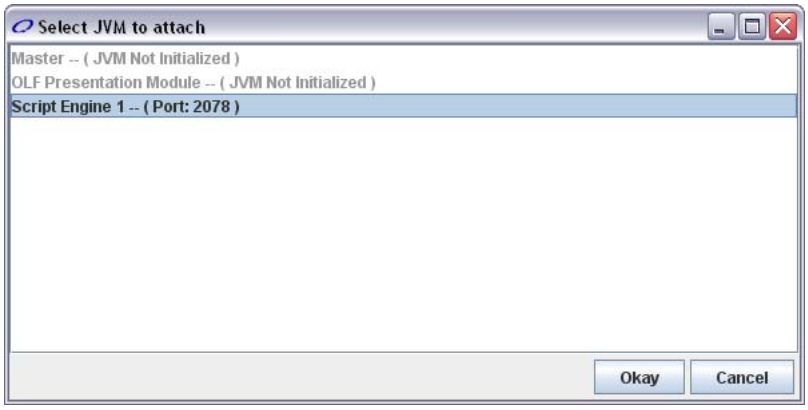


8.7.3 Run/Debug the Plugin

1. Go back to the Java IDE to attach the JVM and start debugging.

Eclipse	Select **OpenJVS→Debug Local Project**.
NetBeans	Select **Debug Local OpenJVS Project** from the project's pop-up menu.

2. Select the port entry matching the JVM Debug Port from the list provided and click **Okay**.



3. Check the Output Console of the IDE. It should display **BUILD SUCCESSFUL**.
4. In OpenLink, load the **SampleScript** task and click **Run Task**.
5. The task will stop at the breakpoint set in Eclipse|NetBeans. Step through the plugin in debug mode.

9.0 CREATE A PLUGIN IN AN EXISTING PROJECT

OpenJVS projects can contain any number of plugins. The steps to create and execute a plugin within an existing project are outlined below:

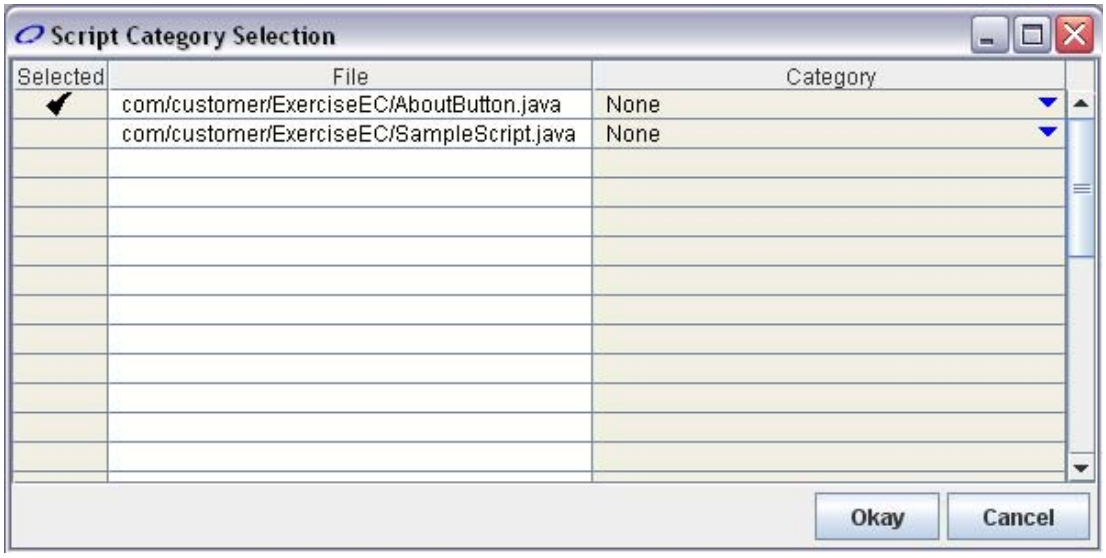
- Create the sample plugin in the Netbeans|Eclipse IDE
- Save the plugin into the OpenLink System
- Save the plugin into a Task
- Execute the Task.

9.1.1 Create the Plugin

1. Launch the Eclipse|NetBeans IDE.
2. Open an existing project and fully expand the **src** section of project.
3. Right click on the SamplePlugin.java file and select **Copy** from the drop down menu.
4. Once you have selected Copy right click on the section or folder containing the source and select **Paste** from the drop down menu.
5. Rename the file "MyNewPlugin".
6. Using the IDE editor, code the plugin appropriately.
7. **Save** the code in the IDE.
8. **Save** the plugin to OpenLink from Eclipse|NetBeans.

Eclipse	Select **OpenJVS→Save Plugin.**
NetBeans	Right click the Project and select **Save OpenJVS Plugins to Endur/Findur**.

9. This will open the *Script Category Selection* window. In this window place a check next to the newly created plugin and select the Okay button.



Note: The Console panel at the bottom of the IDE will provide a listing of the steps being executed and any success or failure messages. When the plugin is saved in The OpenLink System "BUILD SUCCESSFUL" will be displayed in this panel.

9.1.2 Verify the Plugin

In order to verify that successful import of the plugin open the Directory Browser. Under the Directory Name panel, there will be a subdirectory tree that matches the package qualifiers in the plugin. The plugin will be listed in the right-hand panel after clicking on the lowest level directory.

9.1.3 Run the Plugin

1. **Select Script→Task Editor** from the menu bar of the Trading Manager. The Task Editor window will open.
2. Name the task "MyNewTask" within the Task Name field.
3. Right click within the Main Script field to bring up a list of available plugins.
4. Select the plugin that was just imported via the *Script Category Selection* window.
5. Click the **Save** button at the top of the window and close the window. The OpenLink System will automatically load the new task as the current task within the Trading Manager.
6. Click the **Run Task** button.

10.0 ADDITIONAL OpenJVS CAPABILITIES

10.1 Export a Project

This task exports all source and class bytecode for the entire project to the AB_JAVA_WORKSPACE location. It also creates two OpenLink specific files: jproject.classpath and <projectname>.out.

The project directory and the <projectname>.out file will reside directly under the AB_JAVA_WORKSPACE location.

jproject.classpath contains a list of other OpenJVS projects in OpenLink on which this project depends. <projectname>.out contains directives to import the project into another OpenLink installation.

All package directories will be created automatically during this process. The class files will be placed in the same directory as their source file counterparts.

10.1.1Export from the Command Line

openjvs_util –e <OpenJVS project> –w <workspace_dir>

Example: openjvs_util –e MyJVS –w D:\OpenJVS

OpenJVS project	The name of the project as it exists in OpenLink.
workspace_dir	The absolute path to where the project will be exported. This is akin to the role played by AB_JAVA_WORKSPACE while in OpenLink, however there is no default.

If there is a local copy of the project at the intended destination, the system will warn that the local project will be replaced and ask for permission to continue. See the section *Modify a Plugin with OpenLink.*

10.1.2Export from the Java IDE

Using this option in the Java IDE will refresh the entire local workspace project from OpenLink. If there are any locally modified plugins with the same name and package designation as plugins in the OpenLink project, a dialog box will prompt for authorization to overwrite them with the OpenLink copy. This task will also export the class files from OpenLink.

Eclipse	Select **OpenJVS→Load Project**
NetBeans	Right-click on the project for the pop-up menu. Click on **Reload OpenJVS Project from Endur/Findur**

10.1.3Export from OpenLink

The task described in the section *Modify a Plugin with OpenLink* performs a full project export.

10.2 Import a Project

This will import the project MyJVS, its plugins, their compiled classes and any OpenJVS project references into OpenLink. If the MyJVS project did not exist, this command would also create the project.

10.2.1 Import from the Command Line

If all the classes exist for the plugin source files, the task will import them. If one or more classes are missing, the task will automatically compile the project before import. An optional JDK must be installed for compilation. It is also possible to force a compilation of the project even if all the class files exist by specifying the –f (force compile) optional flag.

The primary usage for this command line option is an unattended import of entire compiled projects into OpenLink without the need for a locally installed Java compiler. This is highly beneficial when promoting OpenJVS projects into a production environment.

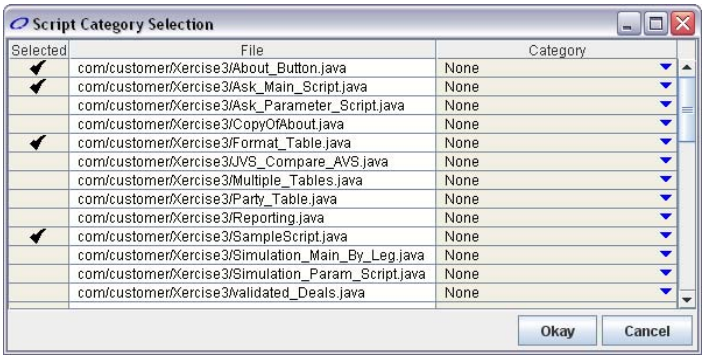
The source data for this option is best obtained from the command line export project capability described in the section *Export a Project*.

openjvs_util –i <absolute path to manifest file> [-f]
Example: openjvs_util –i D:\OpenJVS\MyJVS.out [-f]

10.3 Import a Plugin from the Java IDE

For each plugin being saved (imported into OpenLink), check the Selected column. This will display a check mark indicating the plugin will be saved with this operation. The right-most column is a pick list for selecting one or more categories to which the plugin can belong in OpenLink. The list of categories is built dynamically from the current OpenLink list of categories.

When finished with selecting plugins and categories, click **Okay**.



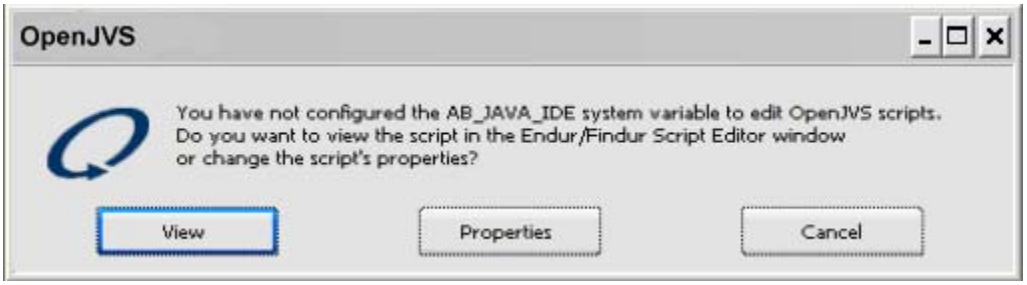
Eclipse	Select **OpenJVS→Save Plugin**.
NetBeans	Right-click on the project for the pop-up menu and click on **Save Plugins to Endur/Findur**

10.4 Modify a Plugin with OpenLink

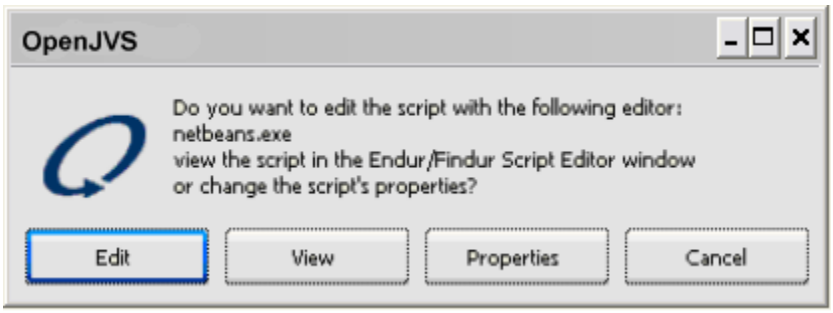
OpenJVS plugin editing can be initiated from OpenLink.

To edit an OpenJVS plugin from OpenLink, double-click on an OpenJVS plugin anywhere it is listed in a panel such as the *Task Editor* referenced in the section *Run a Plugin in OpenLink*. Click on the **Edit** button next to the plugin listed in the Main plugin pick list field.

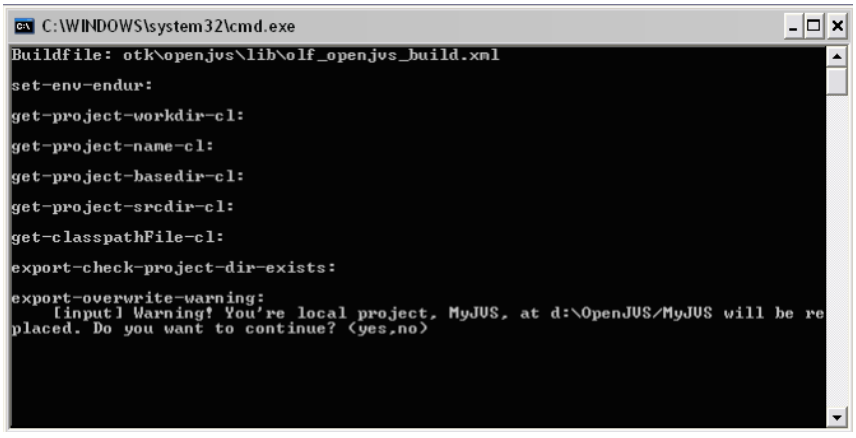
If the AB_JAVA_IDE OpenLink variable is not set, a dialog box displays providing an option to view the plugin in read-only mode in the *OpenLink Plugin Editor* panel.



If the AB_JAVA_IDE variable is set, its value will be displayed in the dialog box as a confirmation.



When Edit is selected on the dialog, a full export of the project to which the plugin belongs will occur. See *Export a Project* for more information. The project will be exported to the AB_JAVA_WORKSPACE location.



A DOS window will appear briefly displaying the export steps being performed.

If there is a local copy of the project at the intended destination, the system will warn that the local project will be replaced and ask for confirmation.

The editor referenced in the AB_JAVA_IDE variable will then be started. If any keywords were used, they will be replaced with the actual values derived from the plugin, project and AB_JAVA_WORKSPACE.

While OpenLink starts the editor referenced in AB_JAVA_IDE, the editor instance is an independent process from OpenLink.

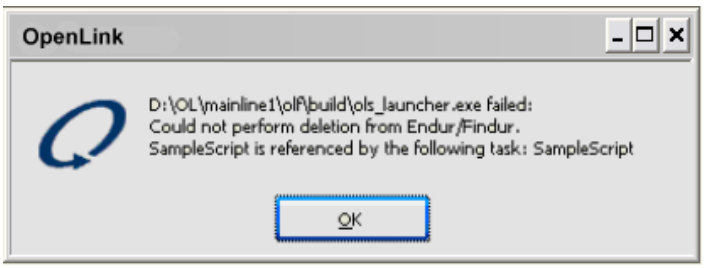
If the editor is already running, the existing instance may be reused or a new one may be launched; this is editor dependent.

10.5 Delete a Project

This only deletes a project from OpenLink. Note the following:

- Any local workspace copies of the project will not be affected.
- Projects referenced by other projects in OpenLink will not be deleted.
- Projects that contain plugins that are referenced by tasks in OpenLink will not be deleted.

For example, try to delete the MyJVS project created in the walkthrough. If the task "SampleScript" still exists and references the "SampleScript" java file in the MyJVS project, the system will present the following dialog:

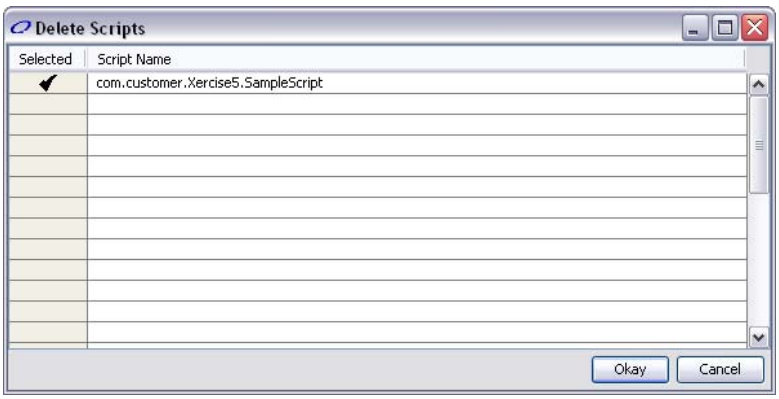


Command line	openjvs_util –d <OpenJVS project name> Example: openjvs_util –d MyJVS
Eclipse	Select **OpenJVS→Delete Project**
NetBeans	Right-click on the project for the pop-up menu and click **Delete OpenJVS Project from OpenLink**.

10.6 Delete a Plugin (IDE Only)

This only deletes plugins from OpenLink. Any local workspace copies of the project will not be affected.

A dialog box will provide a list of all plugins that exist in OpenLink. This is not the list of plugins from the local workspace. Select the targets to delete and click **Okay**.



Plugins referenced by tasks in OpenLink will not be deleted.

Plugins that belong to projects that are referenced by other projects in OpenLink will not be deleted.

Eclipse	Select **OpenJVS→Delete Plugin**
NetBeans	Right-click on the project for the pop-up menu and click **Delete OpenJVS Plugins from Endur/Findur**

10.7 Verify a Deletion from OpenLink

Follow the steps in section *Verify Project Build in OpenLink.*

If a plugin has been deleted, it should no longer exist in the project's subdirectory tree. If the last plugin from a package has been deleted, the package directories will be deleted as well.

If a project has been deleted, the project and all of its contents should no longer exist under the OpenComponent/Site directory.

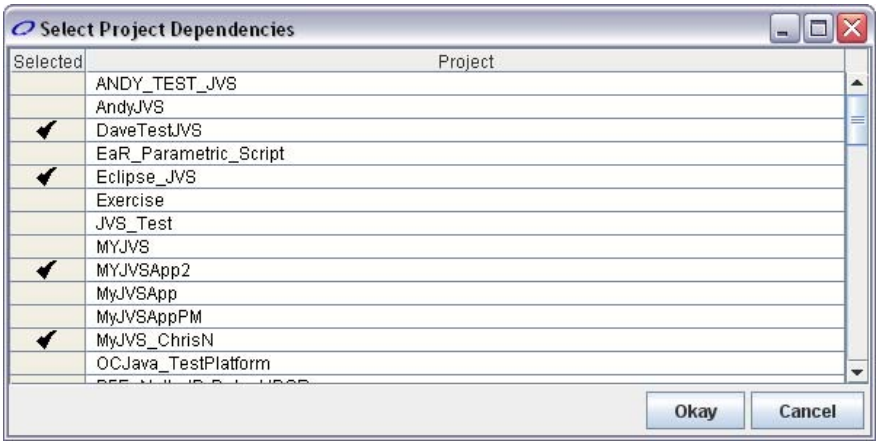
10.8 Updating Project References (IDE Only)

Use this option to specify projects in OpenLink on which the current project depends. An example of these dependencies could be a common plugin library.

A dialog box will provide a list of all OpenJVS projects that exist in OpenLink. Select the ones your project depends or will depend on and click **Okay**.

This task will update the references in OpenLink and also the local project file jproject.classpath.

An error message will be generated if the current project is selected. Projects cannot reference themselves.



Eclipse	Select **OpenJVS→Update Project**.
NetBeans	Right-click on the project for the pop-up menu and click **Update OpenJVS Project References in Endur/Findur**.

10.9 Compile a Project (IDE Only)

To compile a project, the project must exist in OpenLink for compilation to occur.

The project's classpath already contains the OpenJVS jar file. Therefore, compile errors related to the OpenJVS API will generally be highlighted with the IDE's contextual help.

Compiler error messages will be caught and sent back to the IDE's output panel as shown to the right. Note: (the "o" was removed from the oprint() method call).

Since standard Java is permitted in OpenJVS plugins and the OpenJVS compilation includes the Java libraries, standard Java compiler errors will also be caught and sent back to the IDE's output panel.

```
D:\OpenJVS\MyJVS\src\com\customer\MyJVS
\SampleScript.java:9: cannot find symbol
symbol : method print(java.lang.String)
location: class com.olf.openjvs.OConsole
    OConsole.print("Hello World!");
1 error
BUILD FAILED (total time: 3 seconds)
```

Eclipse	Select **OpenJVS→Compile Project**.
NetBeans	Right-click on the project to get the pop-up menu and click **Compile OpenJVS Project**.

10.10 Listing OpenJVS Projects (Command Line Only)

The command shown below provides a simple console list of existing OpenJVS OpenLink projects.

```
openjvs_util.bat –l
```

11.0 OpenJVS LIMITATIONS

11.1 OpenJVS code can be only run from within OpenLink

While plugin development, debugging, and all the plugin management functions provided by the Ant tasks are generally performed outside OpenLink, the actual running of the plugin must be done from within OpenLink. An OpenJVS plugin cannot access any OpenLink functionality or the OpenLink database from a separate process.

11.2 OpenJVS is single-threaded

Multi-threading is not currently supported. The OpenJVS API checks that all code is run from the main OpenLink thread and prevents OpenJVS code in other threads from executing.

11.3 GUI Support

While OpenJVS does not support the use of Swing, ASK window functionality is fully supported.

12.0 `ERROR MESSAGES AND NOTES

12.1 Error Messages

"Unable to locate tools.jar. Expected to find it in …\olf\bin\olf_dependencies\java\jdk1.5.0_12\lib\tools.jar" This is a warning message from the Java classloader that occurs when Ant is executed and JAVA_HOME is not set correctly. Set environment variable, JAVA_HOME, to the JDK root directory.
table. isTableValid() function throws exception on null table: if(tSimRes.isTableValid() == 0) has to be changed to if(tSimRes == null || tSimRes.isTableValid() == 0)

12.2 Additional Notes

OpenComponents	OpenComponents can reference and call OpenLink resident OpenJVS plugins. The OpenComponents Eclipse plugin can also debug OpenLink resident OpenJVS plugins. OpenJVS cannot call, reference or debug OpenComponent code.
Extending the classpath	This feature was added to the OpenLink System on June 19, 2008 for V82R1 and above. Endur/Findur will append the string stored in the AB_CLASSPATH environment variable to its internal Java classpath. The value of the string must be a correctly formed classpath without a leading separator value - semi-colon (;) on Windows; colon (:) on Unix/Linux. For example, Endur/Findur's default classpath is: olf_openjvs.jar;olf_openjvs_enums.jar;mail.jar;activation.jar;olf_openjvs_bootstrap.jar If you set the AB_CLASSPATH environment variable to: log4j.jar;mycustom.jar the resulting OpenLink System classpath for both compilation and runtime will be: olf_openjvs.jar;olf_openjvs_enums.jar;mail.jar;activation.jar; olf_openjvs_bootstrap.jar;log4j.jar;mycustom.jar The CLASSPATH value will always be appended. It cannot be pre-pended.

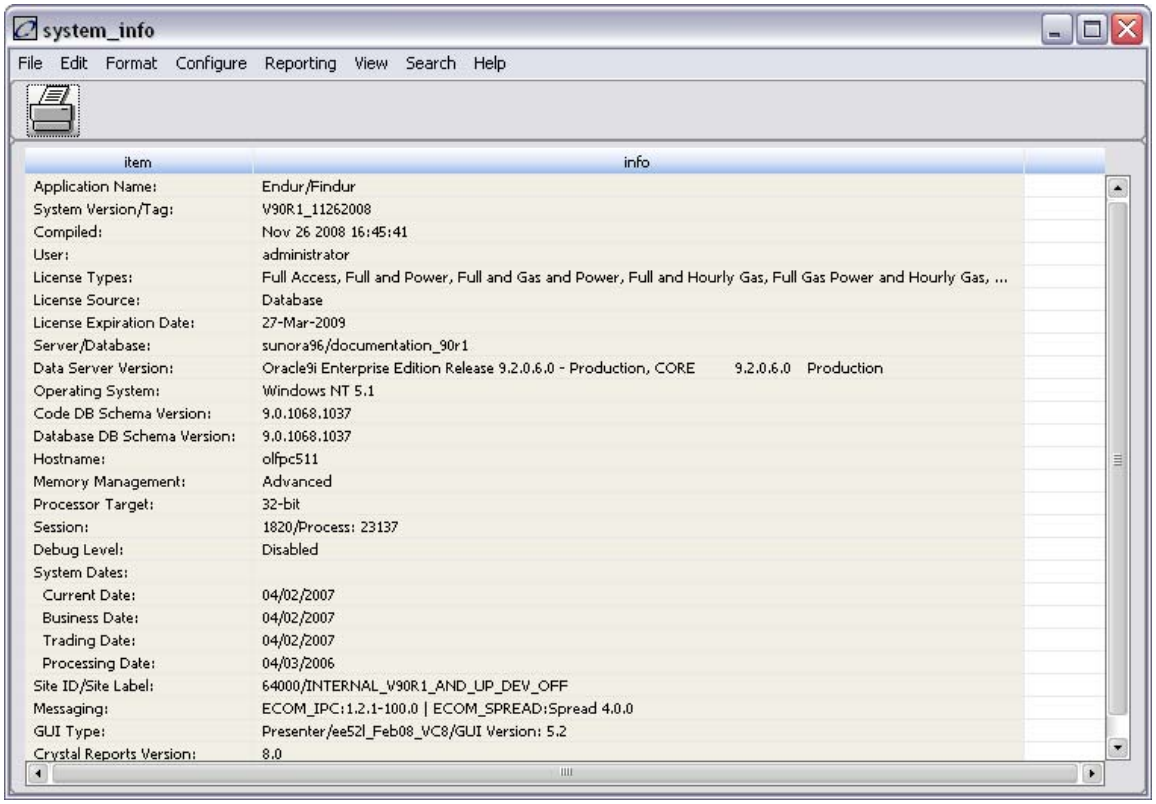
Enumerations	Enumerations need type value to be added: TOOLSET_ENUM.COM_OPT_TOOLSET have to be changed to TOOLSET_ENUM.COM_OPT_TOOLSET.jvsValue()
go to	There is no "go to" statement in java
switch	there is no "switch" statement on string and enumeration in java, can only switch on integer constants (static final int) or literals.
Required change	if(Str.isEmpty(strSimName)) statements have to be changed to if(Str.isEmpty(strSimName) != 0)
Required change	tData.select(tTranRes, "#PFOLIO_RESULT_TYPE.BASE_PV_RESULT(base_mtm), " has to be changed to: table.select(tTranRes, Str.intToStr(PFOLIO_RESULT_TYPE.BASE_PV_RESULT.jvsValue()) + "(base_mtm), "
Required change	m_INCStandard.STD_InitRptMgrConfig(error_log_file) has to be changed to: m_INCStandard.STD_InitRptMgrConfig(error_log_file, argt);
Required change	temp = TABLE_CopyTable(argt.getTable("bunits", 1)) has to be changed to: temp =argt.getTable("bunits", 1).copyTable();

13.0 APPENDIX A - PROGRAMMING EXAMPLES

13.1 Plugin Development Overview – The AboutButton Plugin

This introductory example provides a framework for developing the plugin examples provided in this Appendix.

You will create a memory table, assign the values of the OpenLink "About" button to the table and view the table. The "About" button is accessed via the OpenLink Console and shows information about the system such as System Version, License Type, License Expiration Date etc. The Table class will be used in this plugin to create and populate the memory table (shown below).



In order to create a blank executable plugin you can copy and rename the sample plugin in your Exercise project; to the name of the plugin you wish to create. Creating a blank plugin this way will create the plugin with all the necessary declarations.

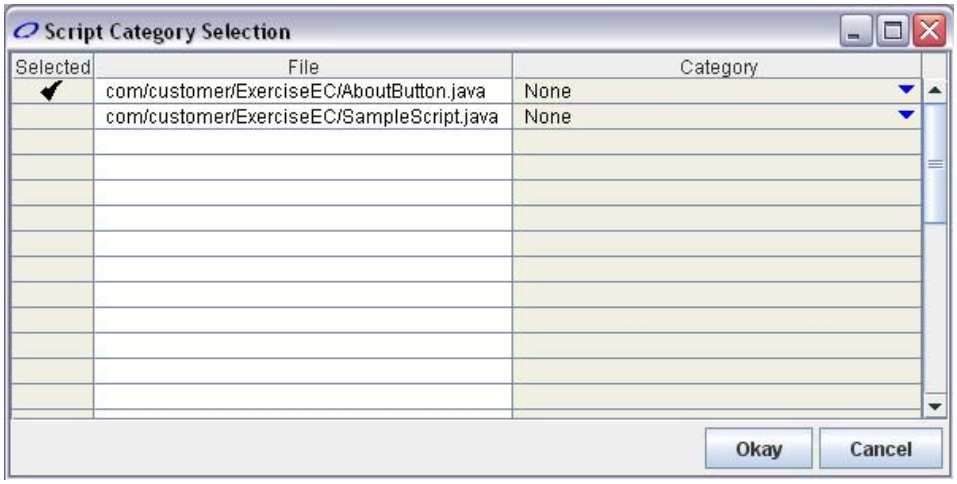
1. Launch the Eclipse|NetBeans IDE.
2. Open the Exercise project and fully expand the **src** section of project.
3. Right click on the SamplePlugin.java file and select **Copy** from the drop down menu.
4. Once you have selected Copy right click on the section or folder containing the source and select **Paste** from the drop down menu.
5. **Rename** the file "AboutButton".
6. In the editor, supply the code shown below.


```
package com.customer.Exercise;

import com.olf.openjvs.*;
import com.olf.openjvs.enums.*;

public class AboutButton implements IScript
{
    public void execute(IContainerContext context) throws OException
    {
    //Declare the table variable
        Table AboutTable;
   //Create the memory table
        AboutTable = Table.tableNew();
    //Set the memory table
        AboutTable = SystemUtil.getInfo();
    //View the memory table
        AboutTable.viewTable();
    //Delete the variable table
        AboutTable.destroy();
    }
}
```

7. **Save** the code in the IDE.
8. **Save** the Plugin to OpenLink from Eclipse|NetBeans.

Eclipse	Select **OpenJVS→Save Plugin.**
NetBeans	Right click the Project and select **Save OpenJVS Plugins to Endur/Findur**.

9. This will open the *Script Category Selection* window. In this window place a check next to the AboutButton plugin and click the **Okay** button.



Note: The Console panel at the bottom of the IDE will provide a listing of the steps being executed and any success or failure messages. When the plugin is saved in The OpenLink System "BUILD SUCCESSFUL" will be displayed in this panel.

13.1.1 Verify the Plugin

In order to verify that successful import of the plugin open the Directory Browser. Under the Directory Name panel, there will be a subdirectory tree that matches the package qualifiers in the plugin. The plugin will be listed in the right-hand panel after clicking on the lowest level directory.

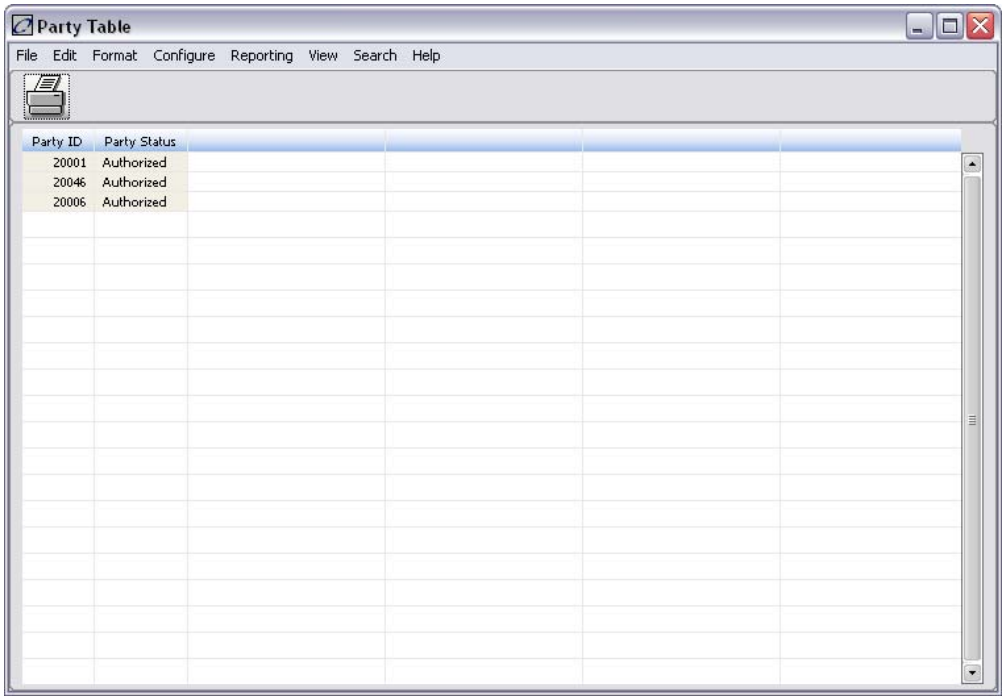
13.1.2 Run the plugin in The OpenLink System

1. **Select Script→Task Editor** from the menu bar of the Trading Manager. The Task Editor window will open.
2. Name the task AboutButton within the Task Name field.
3. Right click within the Main Script field to bring up a list of available plugins.
4. Select the AboutButton plugin that was just imported via the *Script Category Selection* window.
5. Click the **Save** button at the top of the window and close the window. The OpenLink System will automatically load the AboutButton task as the current task within the Trading Manager.
6. Click the **Run Task** button. Two events will occur – the Status will change to 'Succeeded' in the Status field and a memory table containing the About Button information will display.

13.2 Table Creation Plugin

A memory table is a temporary table that exists in memory for the current session. A memory table is an array-like structure used in OpenJVS to hold tabular information in memory. Create memory tables or access memory table data by executing a method. Numerous OpenJVS library methods manipulate data within a memory table.

Memory tables are constructed using columns and rows. However, unlike an array, a memory table may contain a different data type for each column. These data types include string, integer and double. Sub tables can also be added to a memory table.



The memory resources of the workstation running the plugin limit memory available to OpenJVS. Memory tables created in a plugin use memory during the execution of the plugin. Two methods for managing memory usage in a plugin are:

1. Destroy all memory tables at the end of the plugin.
2. Destroy each memory table as soon as it is no longer needed in the program.

Choose either method after considering that Method 1 provides for easy code maintenance and debugging, while Method 2 provides performance advantages for longer plugins.

The Table class will be used in the sample plugin that follows to create and populate a memory table.

13.2.1 Table Creation Code

```
//OpenJVS Create Table

package com.customer.My_Project;
import com.olf.openjvs.*;
import com.olf.openjvs.enums.*;
public class Party_Table implements IScript
{
    public void execute(IContainerContext context) throws OException
    {
        Table TblTest;
        TblTest = Table.tableNew ("Party Table");
        TblTest.addCol( "Party ID",COL_TYPE_ENUM.COL_INT);
        TblTest.addCol( "Party Status",COL_TYPE_ENUM.COL_STRING);
        TblTest.addNumRows(3);
        TblTest.setInt("Party ID", 1, 20001);
        TblTest.setInt("Party ID", 2, 20046);
        TblTest.setInt("Party ID", 3, 20006);
        TblTest.setString( "Party Status", 1, "Authorized");
        TblTest.setString( "Party Status", 2, "Authorized");
        TblTest.setString( "Party Status", 3, "Authorized");
        TblTest.viewTable();
        TblTest.destroy();
    }
}
```

13.3 Database Query Plugin

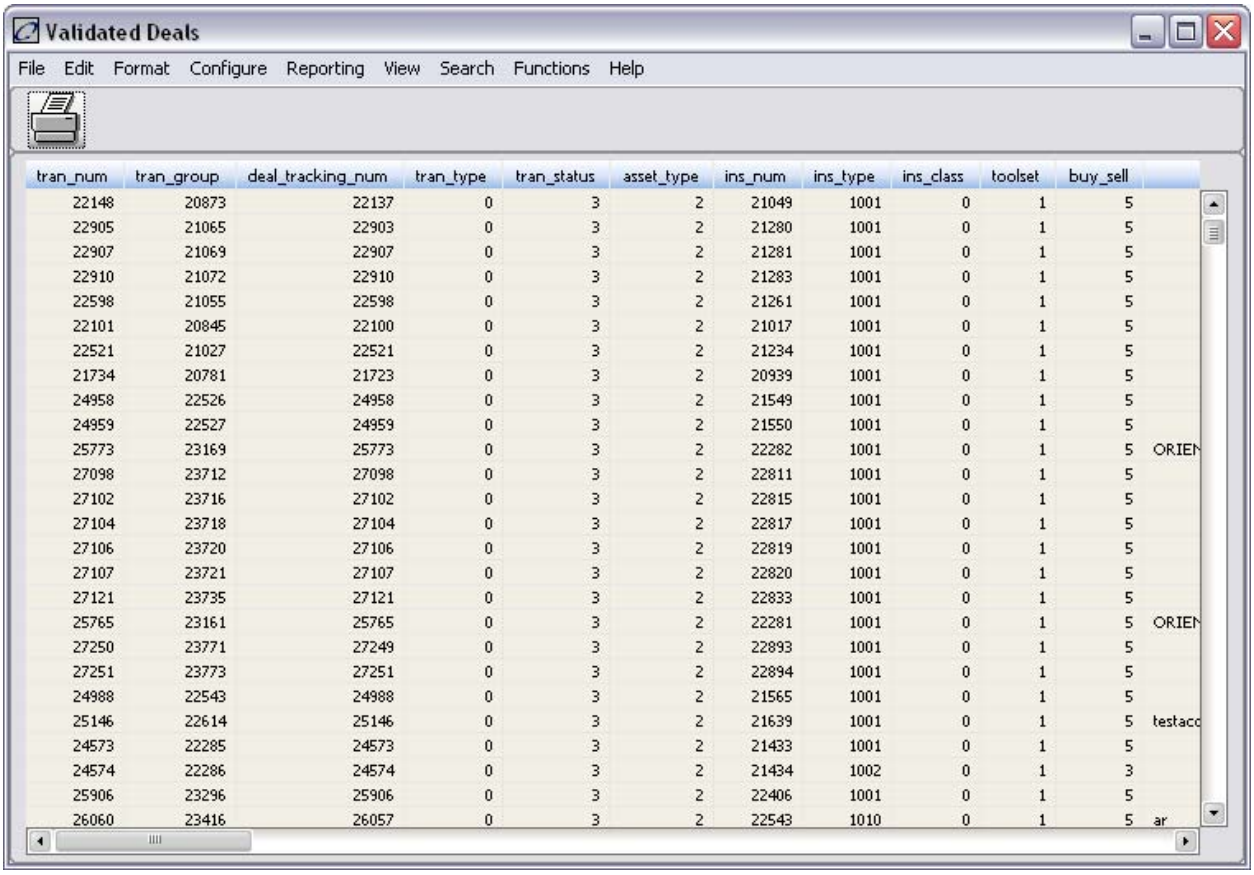
OpenJVS gives the user the ability to write plugins that will execute a query using SQL. The data that is returned can then be formatted and viewed in a table or report.

One of the most common functions of a plugin is to extract data from the one or more the tables within the OpenLink System. OpenJVS provides the developer with various methods that are generally found in the DBaseTable class and Dbase class.

The data extracted from the database tables can be used to print to the console, populate a pick list, generate a memory table, or run a report or simulation. Keep in mind that SQL cannot be directly implemented in a plugins The SQL statement (what, from, where) should be written as a string, enclosed in quotations and insert it as an argument for the method.

13.3.1 Single Table Query

The following plugin extracts data from the ab_tran table and display it within a Table Viewer. It display all deals that have a transaction status of "validated".



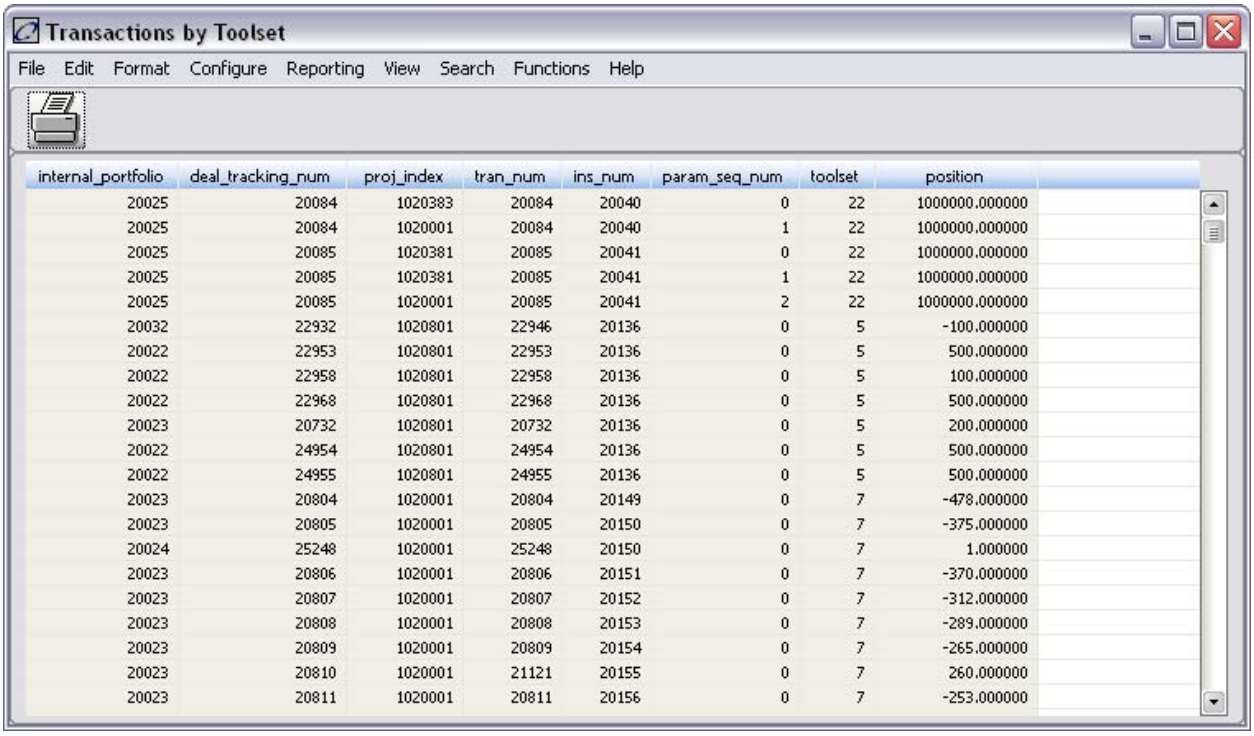
tran_num	tran_group	deal_tracking_num	tran_type	tran_status	asset_type	ins_num	ins_type	ins_class	toolset	buy_sell	
22148	20873	22137	0	3	2	21049	1001	0	1	5	
22905	21065	22903	0	3	2	21280	1001	0	1	5	
22907	21069	22907	0	3	2	21281	1001	0	1	5	
22910	21072	22910	0	3	2	21283	1001	0	1	5	
22598	21055	22598	0	3	2	21261	1001	0	1	5	
22101	20845	22100	0	3	2	21017	1001	0	1	5	
22521	21027	22521	0	3	2	21234	1001	0	1	5	
21734	20781	21723	0	3	2	20939	1001	0	1	5	
24958	22526	24958	0	3	2	21549	1001	0	1	5	
24959	22527	24959	0	3	2	21550	1001	0	1	5	
25773	23169	25773	0	3	2	22282	1001	0	1	5	ORIEN
27098	23712	27098	0	3	2	22811	1001	0	1	5	
27102	23716	27102	0	3	2	22815	1001	0	1	5	
27104	23718	27104	0	3	2	22817	1001	0	1	5	
27106	23720	27106	0	3	2	22819	1001	0	1	5	
27107	23721	27107	0	3	2	22820	1001	0	1	5	
27121	23735	27121	0	3	2	22833	1001	0	1	5	
25765	23161	25765	0	3	2	22281	1001	0	1	5	ORIEN
27250	23771	27249	0	3	2	22893	1001	0	1	5	
27251	23773	27251	0	3	2	22894	1001	0	1	5	
24988	22543	24988	0	3	2	21565	1001	0	1	5	
25146	22614	25146	0	3	2	21639	1001	0	1	5	testacc
24573	22285	24573	0	3	2	21433	1001	0	1	5	
24574	22286	24574	0	3	2	21434	1002	0	1	3	
25906	23296	25906	0	3	2	22406	1001	0	1	5	
26060	23416	26057	0	3	2	22543	1010	0	1	5	ar

13.3.2Single Table Query Code

```
//OpenJVS Single Table Query
package com.customer. My_Project;
import com.olf.openjvs.*;
public class Validated_Deals implements IScript
{
    public void execute(IContainerContext context) throws OException
    {
        Table TblTest;
        TblTest = Table.tableNew ("Validated Deals");
        DBase.runSql("select * from ab_tran where tran_status = 3");
        DBase.createTableOfQueryResults(TblTest);
        TblTest.viewTable();
        TblTest.destroy();
    }
}
```

13.3.3 Multiple Table Query

The following example displays the tran_num (from the ab_tran table) and the proj_index (from the parameter table) for specific deals.



internal_portfolio	deal_tracking_num	proj_index	tran_num	ins_num	param_seq_num	toolset	position
20025	20084	1020383	20084	20040	0	22	1000000.000000
20025	20084	1020001	20084	20040	1	22	1000000.000000
20025	20085	1020381	20085	20041	0	22	1000000.000000
20025	20085	1020381	20085	20041	1	22	1000000.000000
20025	20085	1020001	20085	20041	2	22	1000000.000000
20032	22932	1020801	22946	20136	0	5	-100.000000
20022	22953	1020801	22953	20136	0	5	500.000000
20022	22958	1020801	22958	20136	0	5	100.000000
20022	22968	1020801	22968	20136	0	5	500.000000
20023	20732	1020801	20732	20136	0	5	200.000000
20022	24954	1020801	24954	20136	0	5	500.000000
20022	24955	1020801	24955	20136	0	5	500.000000
20023	20804	1020001	20804	20149	0	7	-478.000000
20023	20805	1020001	20805	20150	0	7	-375.000000
20024	25248	1020001	25248	20150	0	7	1.000000
20023	20806	1020001	20806	20151	0	7	-370.000000
20023	20807	1020001	20807	20152	0	7	-312.000000
20023	20808	1020001	20808	20153	0	7	-289.000000
20023	20809	1020001	20809	20154	0	7	-265.000000
20023	20810	1020001	21121	20155	0	7	260.000000
20023	20811	1020001	20811	20156	0	7	-253.000000

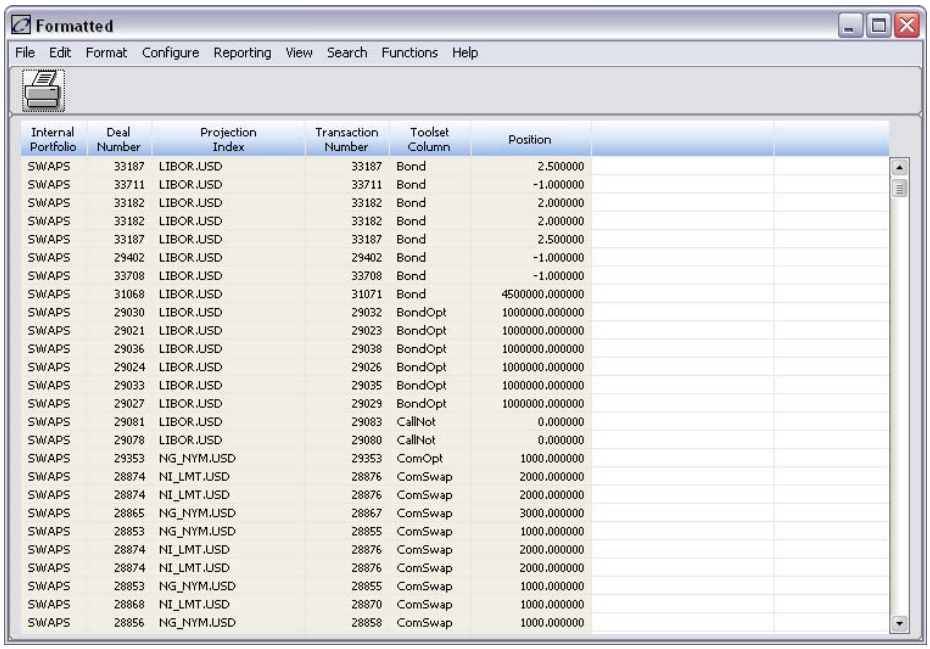
13.3.4 Multiple Table Query Code

```
//OpenJVS Multiple Table Query

package com.customer.My_Project;
import com.olf.openjvs.*;
import com.olf.openjvs.enums.*;
public class Multiple_Tables implements IScript
{
    public void execute(IContainerContext context) throws OException
    {
        Table TblTest;
        String v_what;
        String v_from;
        String v_where;
        TblTest = Table.tableNew ();
        TblTest.setTableTitle("Transactions by Toolset");
     v_what = "a.internal_portfolio"
             +", a.deal_tracking_num"
             +", p.proj_index"
             +", a.tran_num"
             +", a.ins_num"
             +", p.param_seq_num"
             +", a.toolset"
             +", a.position";
        v_from = "ab_tran a, parameter p";
        v_where = "tran_status = 3 AND trade_flag = 1 AND a.ins_num=
p.ins_num";
       DBaseTable.loadFromDbWithSQL(TblTest, v_what, v_from, v_where);
       TblTest.viewTable();
       TblTest.destroy();
    }
}
```

13.4 Formatted Table Plugin

When displaying memory tables, the data can be formatted for use in a display window or in a report. Options include the ability to sort, group, or sum data. There are a number of formatting methods available to do this within the Table class of the Javadoc.

The following example formats the table from the previous example ("Multiple Tables"). Some of the formatting methods that will be introduced in this exercise will hide columns, rename columns and group columns.



Internal Portfolio	Deal Number	Projection Index	Transaction Number	Toolset Column	Position
SWAPS	33187	LIBOR.USD	33187	Bond	2.500000
SWAPS	33711	LIBOR.USD	33711	Bond	-1.000000
SWAPS	33182	LIBOR.USD	33182	Bond	2.000000
SWAPS	33182	LIBOR.USD	33182	Bond	2.000000
SWAPS	33187	LIBOR.USD	33187	Bond	2.500000
SWAPS	29402	LIBOR.USD	29402	Bond	-1.000000
SWAPS	33708	LIBOR.USD	33708	Bond	-1.000000
SWAPS	31068	LIBOR.USD	31071	Bond	4500000.000000
SWAPS	29030	LIBOR.USD	29032	BondOpt	1000000.000000
SWAPS	29021	LIBOR.USD	29023	BondOpt	1000000.000000
SWAPS	29036	LIBOR.USD	29038	BondOpt	1000000.000000
SWAPS	29024	LIBOR.USD	29026	BondOpt	1000000.000000
SWAPS	29033	LIBOR.USD	29035	BondOpt	1000000.000000
SWAPS	29027	LIBOR.USD	29029	BondOpt	1000000.000000
SWAPS	29081	LIBOR.USD	29083	CallNot	0.000000
SWAPS	29078	LIBOR.USD	29080	CallNot	0.000000
SWAPS	29353	NG_NYM.USD	29353	ComOpt	1000.000000
SWAPS	28874	NI_LMT.USD	28876	ComSwap	2000.000000
SWAPS	28874	NI_LMT.USD	28876	ComSwap	2000.000000
SWAPS	28865	NG_NYM.USD	28867	ComSwap	3000.000000
SWAPS	28853	NG_NYM.USD	28855	ComSwap	1000.000000
SWAPS	28874	NI_LMT.USD	28876	ComSwap	2000.000000
SWAPS	28874	NI_LMT.USD	28876	ComSwap	2000.000000
SWAPS	28853	NG_NYM.USD	28855	ComSwap	1000.000000
SWAPS	28868	NI_LMT.USD	28870	ComSwap	1000.000000
SWAPS	28856	NG_NYM.USD	28858	ComSwap	1000.000000

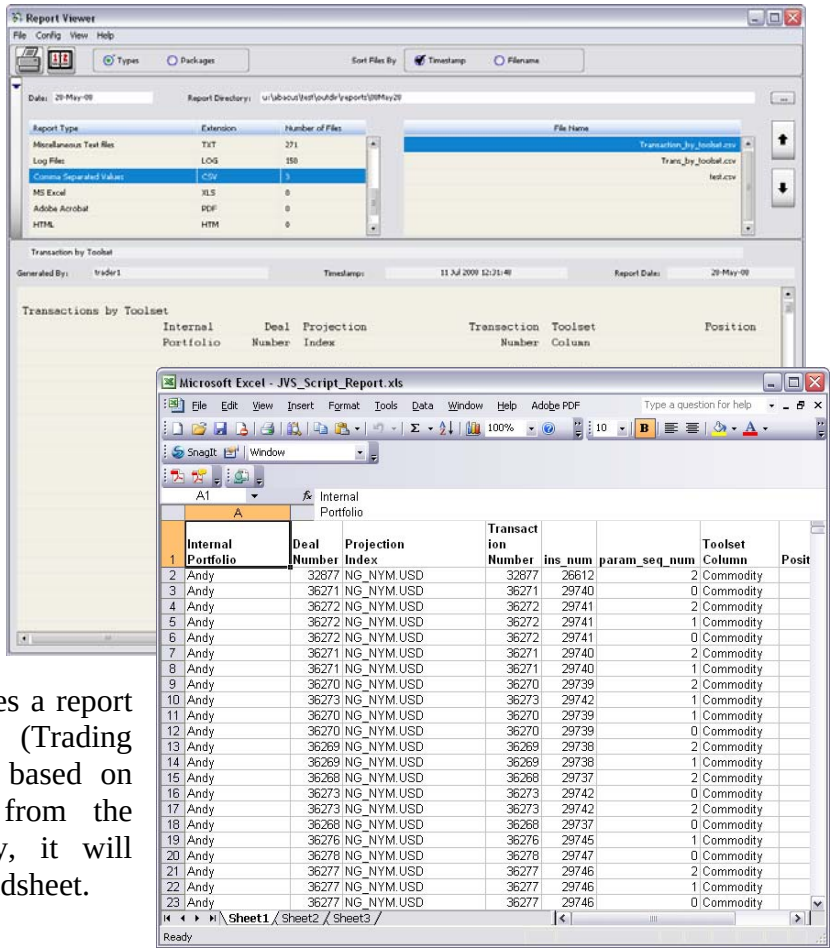
13.4.1 Formatted Tables Code

```
//OpenJVS Format Table
package com.customer.MyProject;
import com.olf.openjvs.*;
import com.olf.openjvs.enums.*;
public class Format_Table implements IScript
{
    public void execute(IContainerContext context) throws OException
    {
        Table TblTest;
        //Declaring variables
        String v_what;
        String v_from;
        String v_where;
        //Create memory table
        TblTest = Table.tableNew ();
        TblTest.setTableTitle("Formatted");
```

```
        //Creating SQL string
        v_what = "a.internal_portfolio"
             +", a.deal_tracking_num"
             +", p.proj_index"
             +", a.tran_num"
             +", a.ins_num"
             +", p.param_seq_num"
             +", a.toolset"
             +", a.position";
        v_from = "ab_tran a, parameter p";
        v_where = "tran_status = 3 AND trade_flag = 1 AND a.ins_num= p.ins_num";
        //Executing SQL string
       int Ret_Val = DBaseTable.loadFromDbWithSQL(TblTest, v_what, v_from, v_where);
        //Formatting
        if (Ret_Val != 1)
        {
               OConsole.oprint("SQL FAILED!");
               TblTest.destroy();
        }
        else
        {
       TblTest.colHide("param_seq_num");
        TblTest.colHide("ins_num");
        TblTest.setColFormatAsRef("toolset", SHM_USR_TABLES_ENUM.TOOLSETS_TABLE);
        TblTest.setColFormatAsRef("proj_index", SHM_USR_TABLES_ENUM.INDEX_TABLE );
        TblTest.setColFormatAsRef("tran_status", SHM_USR_TABLES_ENUM.TRANS_STATUS_TABLE );
        TblTest.setColFormatAsRef("internal_portfolio", SHM_USR_TABLES_ENUM.PORTFOLIO_TABLE );
        TblTest.convertColToString(TblTest.getColNum("toolset"));
        TblTest.convertColToString(TblTest.getColNum("proj_index"));
        TblTest.group("internal_portfolio, toolset");
        TblTest.setColTitle("internal_portfolio", "Internal\nPortfolio");
        TblTest.setColTitle("deal_tracking_num", "Deal\nNumber");
        TblTest.setColTitle("proj_index", "Projection\nIndex");
        TblTest.setColTitle("tran_num", "Transaction\nNumber");
        TblTest.setColTitle("toolset", "Toolset\nColumn");
        TblTest.setColTitle("position", "Position");
        TblTest.viewTable();
        TblTest.destroy();
        }
    }
}
```

13.5 Reporting Plugin

OpenLink provides multiple options to output data to a report file. Within the system itself, output data can be directed to the Report Viewer. The Report Viewer is accessible from the Trading Manager be selecting **View →Report Viewer** or via the Report Viewer icon from the Operations Manager.



In addition to the Report Viewer and its reporting functionality, data can be exported to other third-party reporting applications such as Crystal Reports, MS Excel, or Lotus Notes.

The following example generates a report within the Report Viewer (Trading manager/View/Report Viewer) based on the "Format Table" plugin from the previous exercise Additionally, it will export the data to an Excel spreadsheet.

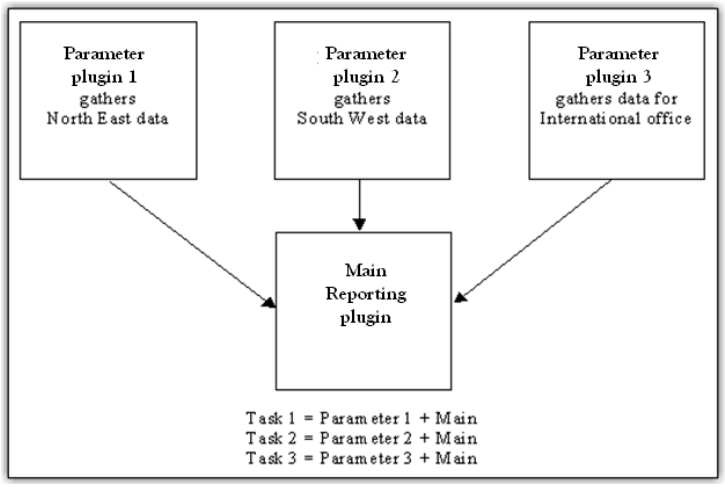
13.5.1 Reporting Code

```
//Reporting Code inserted between the highlighted lines in the Format Table Program
TblTest.viewTable();
Report.reportStart("Transaction_by_toolset.csv", "Transaction by Toolset");
TblTest.formatSetWidth("trans_status", 20);
TblTest.formatSetWidth("tran_num", 20);
TblTest.formatSetJustifyLeft("trans_num");
TblTest.formatSetWidth("ins_num", 18);
Report.printTableToReport(TblTest,REPORT_ADD_ENUM.FIRST_PAGE);
Report.reportEnd();
TblTest.excelSave("c:\\JVS_Script_Report.xls", "Sheet1", "A1");
//New Reporting Code inserted above this Point in the Format Table Program
TblTest.destroy();
```

13.6 Parameter Plugin – User Input

It is often necessary to ask users for input for querying purposes. OpenLink has created the concept of tasks, (see the section *How OpenJVS Works)* to simplify and standardize plugin architecture. Each task consists of one to four plugins (Parameter, Main, Page Viewer, Output); each designed to perform various functions.

Typically, a Parameter plugin is used to pass data to a Main plugin. The programmer has access to a global memory table called the arguments table that is allocated by the system before the execution of the parameter plugin and is passed to the Main plugin. This global memory table can be accessed using a method named getArgumentsTable (). The programmer uses methods like AddCol, AddRow, and Set methods to fill the table with data for use in the Main plugin. Parameter data can be hard coded, read from an external source, or input via a user interface. This allows the user to create variations of the same report without having to maintain multiple copies of the same main plugin. For example, running parameter plugins specific for a geographical region, followed by a main plugin that summarizes and displays the data results in a series of regional reports, as illustrated below.



The Main plugin is used to process data. It can receive parameters via the argument global table, and use that information to create a report, write data to a table, etc. It can also be coded to fill a global table named returnt that passes information to an Output plugin that runs after the Main plugin successfully completes. The returnt memory table can be accessed by using a method named getReturnTable (). The returnt table populated by the Main plugin is passed to the Output plugin.

Table data is typically passed between plugins in arguments table that holds the data that will be used by another plugin. It is overwritten each time it is used and should not be destroyed at the end of each use by the plugin. The arguments table is used by the SCRENG and cannot pass data if the plugin is not run by the SCRENG. When using plugins (i.e., Output plugins) that are run by the Trading Manager, returnt is the preferred method for passing data outside of the SCRENG. Most memory tables used in OpenJVS are created and destroyed within the same source code file (i.e., function as variables with local scope).

In summary, the arguments table allows the OpenJVS programmer to pass data between Parameter plugins and Main plugins. It is used to facilitate a modular approach to reporting and processing that uses combinations of many Parameter plugins and only a few Main plugins. This approach is suggested to promote code reutilization. The returnt table allows the OpenJVS programmer to pass data to other applications outside of the SCRENG process. The Crystal Reports interface uses returnt table to pass a memory table with data to a previously formatted Crystal Report.

13.6.1 The Task Editor

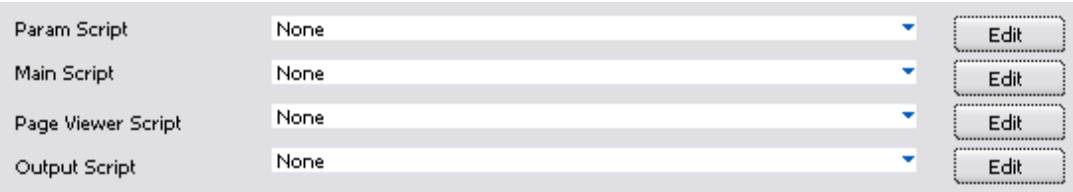
SCRENG runs each plugin from the OpenLink Task Editor in the order shown below.

1. Parameter Scripts (Plugins)
2. Page Viewer Scripts (Plugins)
3. Main Scripts (Plugins)
4. Output Scripts (Plugins)

The SCRENG does not have to be running for these plugins to execute. However, Output plugins rely on the Main plugin (that does rely on the SCRENG) to run. Output plugins should not be run alone.

Parameter plugins are run by the SCRENG prior to the execution of the Main plugin so that parameters are available for use by the Main plugin. Page Viewer plugins must be executed prior to the Main plugin because the Display Table (Page Viewer) must be running in order for it to be populated by data from the Main plugin.

While plugin types (param, main, output,…) are not supported in OpenJVS, the Task Editor is where plugins can be assigned to run a particular task.



13.6.2 Ask Functionality

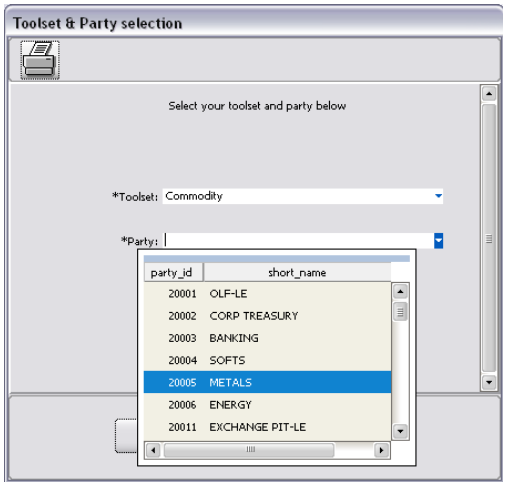
The Ask class allows for the creation of data entry screens within the Parameter plugins. These data entry screens can allow the end user to input data. For example, the developer can write a plugin that will display a data entry screen allowing the user to input a value, such as a date. This value can then be used in some way within the main plugin.

The Ask class has methods that will also allow for the creation of data entry screens where a user can select their choice from a pick list. For example, the developer can define a pick list (ex. toolset) within the Ask method. Once the plugin is run, the data entry screen would appear with the toolset pick list available.

13.6.3 Ask Functionality Plugin

The following example uses several methods from the Ask class in order to allow the user to choose a toolset and an internal party from a pick list. This will require a pick list for Toolsets, and a pick list for Parties.

Once these pick list are created, they will be used within the methods of the ASK class.



13.6.4 Parameter Plugin Output

Once the parameter plugin is run, a data entry screen will appear. The user will be required to select a "Toolset" and a "Party".

13.6.5 Parameter Plugin Code

```
//OpenJVS Ask_Functionality_Script
package com.customer.MyProject;
import com.olf.openjvs.*;
import com.olf.openjvs.enums.*;
/*
 Description: This exercise will allow the user to create a data entry screen. This
 script will serve as a parameter script.
 1. Create tables
2. Create a dialog box
3. Retrieve return values
4. Populate argt with return values
5. Destroy tables
*/
public class Ask_Parameter_Script implements IScript
{
    public void execute(IContainerContext context) throws OException
    {

        //table pointers
         Table tToolset, tParty, tToolsetSelected;
         Table tAsk, tPartySelected ;



         //integer variables:
         int tRet, pRet;
         int iToolset, iParty;
```

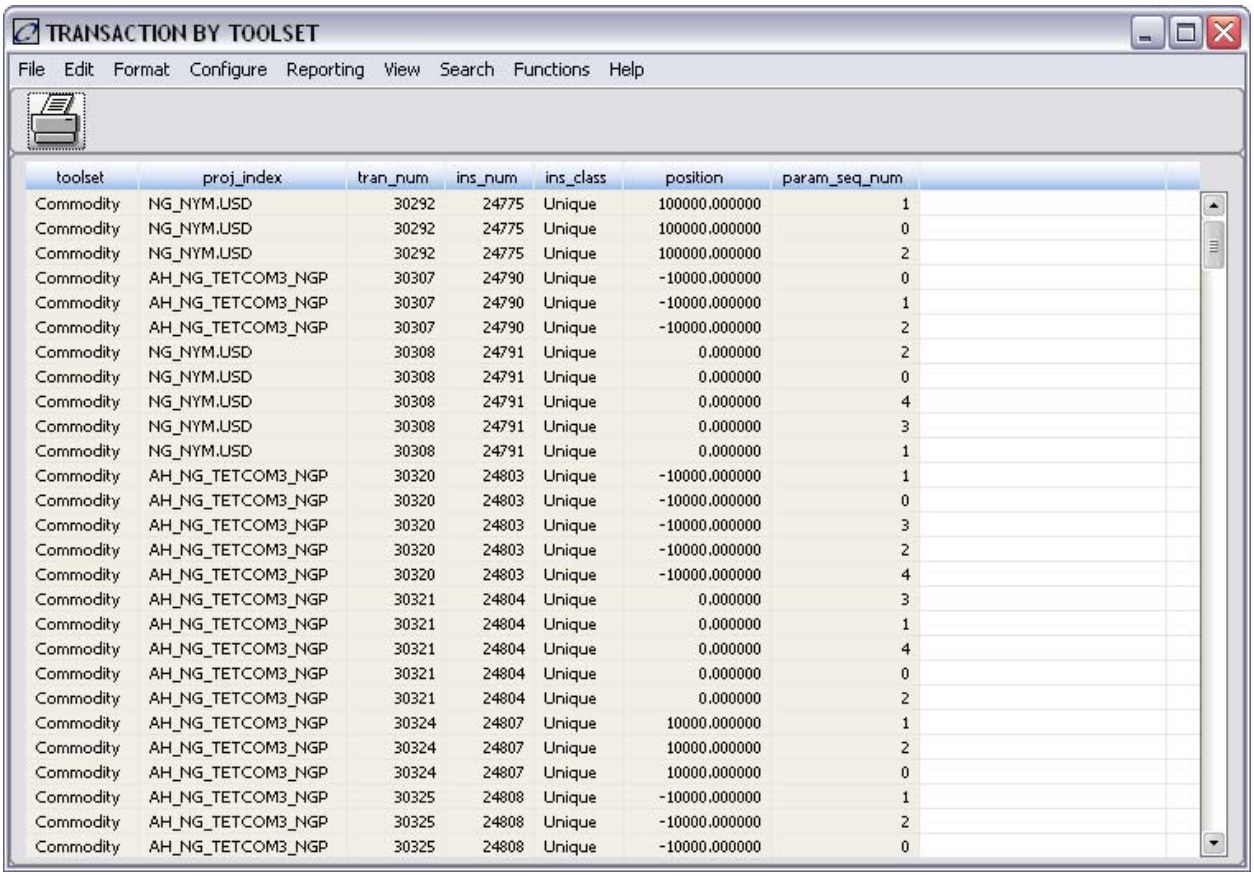
```
        //string variables
        String tWhat, tFrom, tWhere; //toolset query
        String pWhat, pFrom, pWhere; //party query
        //create tables
        tToolset = Table.tableNew("Toolset Listing");
        tParty = Table.tableNew("Party Listing");
        tAsk = Table.tableNew();
        //define toolset and party query
        tWhat = "id_number, name";
        tFrom = "toolsets";
        tWhere = "id_number>0";
        pWhat = "party_id, short_name";
        pFrom = "party";
        pWhere = "party_id>0 AND party_status=1";
        //executing sql for the toolset and party tables
        tRet =   DBaseTable.loadFromDbWithSQL(tToolset, tWhat, tFrom, tWhere);
        pRet =   DBaseTable.loadFromDbWithSQL(tParty, pWhat, pFrom, pWhere);
        tToolset.groupFormatted("name");
        //create dialog boxes
        Ask.setAvsTable(tAsk, tToolset, "Toolset", 2,
        ASK_SELECT_TYPES.ASK_SINGLE_SELECT.jvsValue(),1);
        Ask.setAvsTable(tAsk, tParty, "Party", 2,
        ASK_SELECT_TYPES.ASK_SINGLE_SELECT.jvsValue(),1);
        Ask.viewTable(tAsk,"Toolset & Party selection", "Select your toolset and party below");
        tToolsetSelected = tAsk.getTable("return_value",1);
        tPartySelected = tAsk.getTable("return_value",2);
        iToolset = tToolsetSelected.getInt("return_val",1);
        iParty = tPartySelected.getInt("return_val",1);
        //format argt table
        Table argTable = context.getArgumentsTable();
        argTable.addCol("toolset",COL_TYPE_ENUM.COL_INT);
        argTable.addCol("party",COL_TYPE_ENUM.COL_INT);
        argTable.addRow();
        argTable.setInt("toolset", 1, iToolset);
        argTable.setInt("party", 1, iParty);
        tToolset.destroy();
        tParty.destroy();
        tAsk.destroy();

    }
}
```

13.6.6 Main Processing Plugin

Within the main plugin of this task, the selections made in the parameter plugin (toolset and party selection) will be processed. These selections will then be used within the SQL statement of the main plugin to produce the table output.

13.6.7 Main Processing Plugin Output

The output of this screen will be a memory table that will display information about the selected toolset and party.



toolset	proj_index	tran_num	ins_num	ins_class	position	param_seq_num
Commodity	NG_NYM.USD	30292	24775	Unique	100000.000000	1
Commodity	NG_NYM.USD	30292	24775	Unique	100000.000000	0
Commodity	NG_NYM.USD	30292	24775	Unique	100000.000000	2
Commodity	AH_NG_TETCOM3_NGP	30307	24790	Unique	-10000.000000	0
Commodity	AH_NG_TETCOM3_NGP	30307	24790	Unique	-10000.000000	1
Commodity	AH_NG_TETCOM3_NGP	30307	24790	Unique	-10000.000000	2
Commodity	NG_NYM.USD	30308	24791	Unique	0.000000	2
Commodity	NG_NYM.USD	30308	24791	Unique	0.000000	0
Commodity	NG_NYM.USD	30308	24791	Unique	0.000000	4
Commodity	NG_NYM.USD	30308	24791	Unique	0.000000	3
Commodity	NG_NYM.USD	30308	24791	Unique	0.000000	1
Commodity	AH_NG_TETCOM3_NGP	30320	24803	Unique	-10000.000000	1
Commodity	AH_NG_TETCOM3_NGP	30320	24803	Unique	-10000.000000	0
Commodity	AH_NG_TETCOM3_NGP	30320	24803	Unique	-10000.000000	3
Commodity	AH_NG_TETCOM3_NGP	30320	24803	Unique	-10000.000000	2
Commodity	AH_NG_TETCOM3_NGP	30320	24803	Unique	-10000.000000	4
Commodity	AH_NG_TETCOM3_NGP	30321	24804	Unique	0.000000	3
Commodity	AH_NG_TETCOM3_NGP	30321	24804	Unique	0.000000	1
Commodity	AH_NG_TETCOM3_NGP	30321	24804	Unique	0.000000	4
Commodity	AH_NG_TETCOM3_NGP	30321	24804	Unique	0.000000	0
Commodity	AH_NG_TETCOM3_NGP	30321	24804	Unique	0.000000	2
Commodity	AH_NG_TETCOM3_NGP	30324	24807	Unique	10000.000000	1
Commodity	AH_NG_TETCOM3_NGP	30324	24807	Unique	10000.000000	2
Commodity	AH_NG_TETCOM3_NGP	30324	24807	Unique	10000.000000	0
Commodity	AH_NG_TETCOM3_NGP	30325	24808	Unique	-10000.000000	1
Commodity	AH_NG_TETCOM3_NGP	30325	24808	Unique	-10000.000000	2
Commodity	AH_NG_TETCOM3_NGP	30325	24808	Unique	-10000.000000	0

13.6.8 Main Processing Plugin Code

```
//OpenJVS Ask Functionality Main Script
package com.customer.MyProject;
import com.olf.openjvs.*;
import com.olf.openjvs.enums.*;
/*
 Description: This script will serve as the Main script of the task.
    The return values from the paramter script will be formatted and
    a report will be created.
```

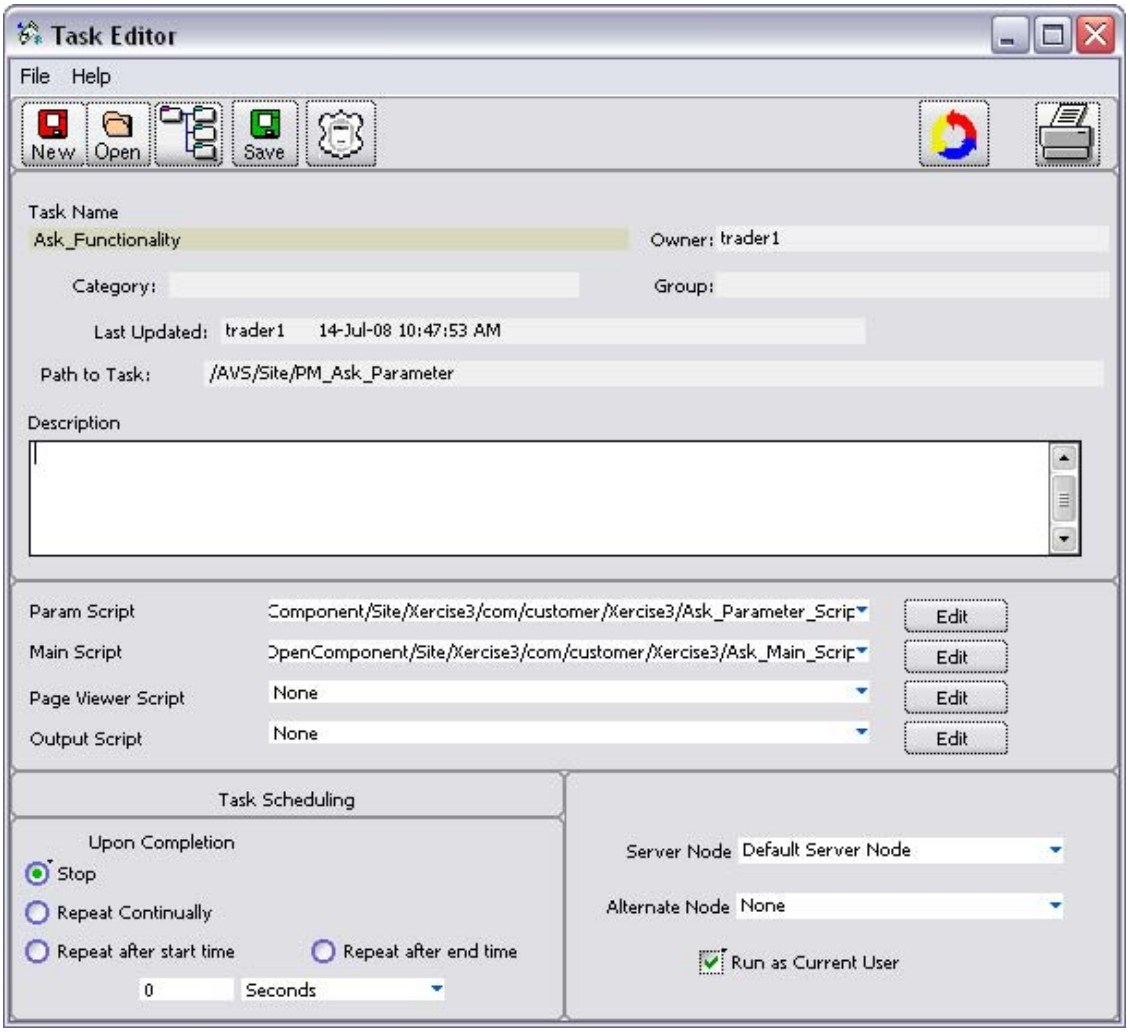
```
STEPS:
 1. Retrieve the
values from the arguments table:
2. Create a Table
3. Format the table
4. View the table
 */
public class Ask_Main_Script implements IScript
{
    public void execute(IContainerContext context) throws OException
    {
        Table tDeals;

        int iToolset, iParty, iRetval;
         String sWhat, sFrom, sWhere;
       Table argTable = context.getArgumentsTable();
       iToolset = argTable.getInt("toolset",1);
       iParty = argTable.getInt("party",1);
       sWhat = "a.toolset"
             +", p.proj_index"
             +", a.tran_num"
             +", a.tran_status"
             +", a.ins_num"
             +", a.ins_class"
             +", a.position"
             +", p.param_seq_num";
       sFrom = "ab_tran a, parameter p";
       sWhere = "tran_status=3 AND trade_flag=1 " +
               " AND toolset= "+ iToolset +
               " AND internal_bunit= "+ iParty +
               " AND a.ins_num = p.ins_num";
       tDeals = Table.tableNew("TRANSACTION BY TOOLSET");
       iRetval = DBaseTable.loadFromDbWithSQL(tDeals, sWhat, sFrom, sWhere);
       if (iRetval != 1)
       {
           OConsole.oprint("SQL Failed!");
       }
       else
       {
```

```
            tDeals.colHide("tran_status");
            tDeals.defaultFormat();
            tDeals.setColTitle("toolset","Toolset");
            tDeals.convertColToString(tDeals.getColNum("toolset"));
            tDeals.convertColToString(tDeals.getColNum("proj_index"));
            tDeals.groupFormatted("tran_num");

            tDeals.viewTable();

            //report

            if (tDeals.getNumRows()>0)
            {
                tDeals.setColTitle("toolset","Toolset");
                tDeals.setColTitle("proj_index","Projected\nIndex");
                tDeals.setColTitle("tran_num","Transaction\nNumber");
                tDeals.formatSetWidth("tran_num", 15);
                tDeals.formatSetJustifyCenter("tran_num");
                tDeals.setColTitle("tran_num","Transaction\nNumber");
                tDeals.setColTitle("ins_num","Instrument\nNumber");
                tDeals.setColTitle("position","Position");
                tDeals.setColTitle("param_seq_num","Param\nSeq\nNum");
                Report.reportStart("Trans_by_toolset.csv","Tran By Toolset");
                Report.printTableToReport(tDeals, REPORT_ADD_ENUM.FIRST_PAGE);
                Report.reportEnd();
                tDeals.excelSave("c:/book1.xls");
            }
            else
            {
                OConsole.oprint("get number of rows error");
            }
            tDeals.destroy();
        }
    }
}
```

13.6.9 Running the Ask Functionality Task

Follow the steps below to run this task in the OpenLink System.

1. Select **AVS→Task Editor** from the menu bar of the Trading Manager. The *Task Editor* window will open.
2. Name the task "Ask_Functionality" within the Task Name field.
3. Right click within the "Param Script" field to bring up available plugins
4. Select the **Ask_Parameter** plugin that was just imported.
5. Right click within the **Main** plugin field to bring up available plugins.
6. Select the **Ask_Main** plugin that was just imported.
7. Click the **Save** button at the top of the window and close the window; the OpenLink System will automatically load the **Ask_Functionality** task as the current task within the Trading Manager.



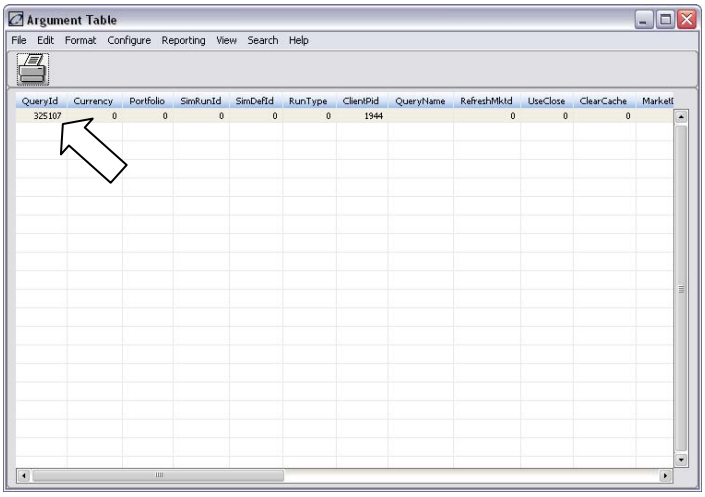
8. Click the **Run Task** button. Two events will occur – the Status will change to "Succeeded" in the Status field and a dialog box will display.

13.7 Query Parameter Plugin

In the previous exercise, a data entry screen was set up to allow users to determine that toolsets and party to use in creating a table of results. Many users of the OpenLink System are already familiar with the query manager, a tool that searches for records based on the criteria set by the user. The information is then displayed in the browser of the Manager.

This example will use a method to assess a user query and process outputs based on the query.

13.7.1 Parameter Plugin Output



When a query is executed within the OpenLink System, a unique query id will be generated for that query. This unique id is temporary. It is only valid for that particular user and session of the OpenLink System. That unique id can be viewed within the AdHoc Query Viewer along with its associated results.

13.7.2 Parameter Plugin Description

The parameter plugin in this exercise will retrieve a Query Id (and the associated results) and populate the arguments global memory table for use within the main plugin.

13.7.3 Parameter Plugin Code

```
//OpenJVS Simulation
package com.customer.MyProject;
import com.olf.openjvs.*;
import com.olf.openjvs.enums.*;
public class Simulation_Param_Script implements IScript
{
    public void execute(IContainerContext context) throws OException
    {
        int iQid;
        String query_name;
        Table argTable = context.getArgumentsTable();
        Sim.createRevalTable(argTable);
        iQid = Query.getQueryIdFromBrowser();
        query_name = Query.getQueryNameFromBrowser();
        OConsole.oprint("*************Query ID:" + iQid);
        OConsole.oprint("*************Query Name:" + query_name);
        argTable.setInt("QueryId", 1, iQid);
        argTable.viewTable();
    }
}
```


13.7.4 Main Processing Plugin Description

The main plugin will generate Simulation Results from the query results (the parameter plugin) that were passed along via the arguments global memory table. Simulation Results are values that the OpenLink System generates for a deal or grouping of deals. These values can include Mark to Market (MTM) and Profit and Loss (PNL). Results can also include information about deals in the system such as unrealized PNL and market sensitivities such as Delta and Gamma.

13.7.5 Main Processing Plugin Code

```
//OpenJVS Main Simulation Script
package com.customer.Xercise3;
import com.olf.openjvs.*;
import com.olf.openjvs.enums.*;
public class Simulation_Main_By_Leg implements IScript
{
    public void execute(IContainerContext context) throws OException
    {
        //declare integer and string variables
        int iQid, iResult;
        String sWhat, sFrom, sWhere;
        //declare table variables
        Table tResultList, tScenResults, tTranResults, tDeals, tOutput;
        //retrieve arguments table
        Table argTable = context.getArgumentsTable();
        //retrieve Query_ID
        iQid = argTable.getInt("QueryId", 1);
        //create results list
        tResultList = Sim.createResultListForSim();
        SimResultType base = SimResultType.create("PFOLIO_RESULT_TYPE.BASE_PV_RESULT");
        SimResult.addResultForSim(tResultList, base);
        //run reval
        tScenResults = Sim.runRevalByQidFixed(argTable, tResultList, Ref.getLocalCurrency());
        tTranResults = SimResult.getTranResults(tScenResults);
        tScenResults.viewTable();
        tTranResults.viewTable();
        //create source table
        tDeals = Table.tableNew("Source Table");
        sWhat = "deal_tracking_num, external_lentity";
        sFrom = "ab_tran, query_result";
        sWhere = "ab_tran.tran_num = query_result.query_result and query_result.unique = " + iQid;
        iResult = DBaseTable.loadFromDbWithSQL(tDeals, sWhat, sFrom, sWhere);
        tDeals.viewTable();
```

```
        //move data from one table to another
        tTranResults.select(tDeals, "external_lentity", "deal_tracking_num EQ $deal_num");
        //create output table
        tOutput = Table.tableNew("SIMULATION");
        tOutput.addCol("external_lentity", COL_TYPE_ENUM.COL_INT);
        tOutput.addCol("deal_num", COL_TYPE_ENUM.COL_INT);
        tOutput.addCol("base_pv", "BASE MTM");
        //sum of pv by deal
        //tOutput.select(tTranResults, "SUM, #BASE_PV_RESULT(base_pv)", "deal_num EQ $deal_num");
        //uncomment for base_pv by leg
        tOutput.select(tTranResults, "deal_leg, #BASE_PV_RESULT(base_pv)", "deal_num EQ $deal_num");
        //format output table
        tOutput.groupFormatted("external_lentity, deal_num, deal_leg");
        tOutput.groupSum("external_lentity");
        tOutput.sum();
        //create report
        Report.reportStart("SIM RESULTS.csv", "SIM RESULTS");
        Report.printTableToReport(tOutput, REPORT_ADD_ENUM.FIRST_PAGE);
        Report.reportEnd();
        tOutput.excelSave("c:\\OpenJVS_Results.xls", "Sheet1", "A1");
        tOutput.viewTable();
        tOutput.destroy();
        tDeals.destroy();
        tResultList.destroy();
        tScenResults.destroy();
        tTranResults.destroy();
    }
}
```