

Introduction.....	2
Prerequisite Environment.....	2
Build Process Overview.....	3
DTP Snapshot from CVS.....	4
Build in Eclipse.....	5
Building the DTP Plug-ins.....	7
Add Exported Plug-ins.....	9
Generate the Source Code Plug-in.....	9
Sanity Check.....	10
Generating the Feature Definitions.....	11
Creating the DTP Download Zips.....	13
Post Build Clean Up.....	14
Appendix One: DTP Code Snapshot for 1.6.....	14
org.eclipse.datatools.build module.....	14
org.eclipse.datatools.connectivity module.....	14
org.eclipse.datatools.enablement module.....	15
org.eclipse.datatools.modelbase module.....	15
org.eclipse.datatools.sqltools module.....	16
Appendix Two: DTP Build Plug-ins.....	16
Appendix Three: Uploading the DTP Builds.....	19

Introduction

This document describes the DTP 1.6 (“Ganymede”) build environment and process. While DTP builds are not as automated as they could be, still DTP is relatively easy to build and anyone familiar with Java, Eclipse (to some extent, Ant) should be able to complete DTP builds without difficulty.

Prerequisite Environment

- **Windows XP:** DTP is currently built on a Windows XP (SP2) machine. In theory DTP can be built on other operation systems, but this has not been tested. Certain issues, like assumptions about path segment separators (“/” or “\”), might cause problems.
- **Sun J2SE 1.4.2:** Again, in theory, DTP could be built using alternative Java platforms, but it is important that any public DTP build use the Sun J2SE 1.4.2, since this is the VM that DTP has declared in it public plans. For convenience, the *bin* folder for Java (containing, for example, *java.exe* and *javac.exe*) should be in your path.
- **Eclipse 3.3.x with EMF 2.2.x and GEF 3.3.x (latest “x”):** As with the Java VM, DTP could be built using other Eclipse platform versions, but this is the published build platform for DTP 1.6. For reasons explained below, you should create an Eclipse workspace with the shortest path possible; for example, directly off the root of a drive using “b” for the directory name. (So, if you load plug-ins into this Eclipse workspace, they appear as immediate children of the “b” directory.)
- **Orbit bundles:** Six 3rd party libraries from the Orbit distribution are required to build DTP. These are included in the Build Tools distribution, in the “orbit” directory, and should be copied into your Eclipse 3.3.x “plugins” folder. At the end of the DTP compile script these libraries will be copied into the DTP download folder for distribution with the DTP build.
- **Apache Ant 1.6** or later should be installed and in your PATH in order to run the DTP ant build files.
- **Winzip:** Is useful for packaging DTP build. Alternatively, you can use zip-compatible compression available from other programs.
- **CVS:** In order for the DTP CVS snapshot script to run, CVS must be installed and in your path.

Build Process Overview

Once the prerequisite environment is in place, the DTP build process consists of the following steps:

1. From the Build Tools distribution, copy the contents of “build_files\step1” to your Eclipse 3.3.x workspace (just copy into the directory, do not import using Eclipse). This gives you two items: a “files” folder containing static content used in the DTP build process, and an Ant build file called “cvsget.xml.” You then run this Ant build file to obtain a code snapshot of DTP from Eclipse CVS.
2. Run Eclipse, and import the plug-in projects into your Eclipse workspace. Build and export a few (details below) plug-ins to a temporary location for later usage.
3. From the Build Tools distribution, copy the contents of “build_files\step2” to your Eclipse 3.3.x workspace, allowing it to overwrite any existing files necessary.
4. From the command line, execute the Ant build script found in your Eclipse workspace under the “org.eclipse.datatools.build” folder. This Ant script, called “build.xml” will assemble all but one of the DTP SDK plug-ins.
5. Returning to the previously exported plug-ins, repack the class files and create jar files for each. Copy (overwrite) them into the DTP build output folder for plug-ins.
6. In the Build Tools distribution, execute a batch file in the “src_plugin” folder to generate the final plug-in needed for the DTP build: the source code plug-in.
7. To check the consistency of the build, in the Build Tools distribution, run the “sanity check” script found in the “sanity” folder. Inspect the output report to make sure that no errors were detected.
8. In the Build Tools distribution, run the update site creation program found in the “site_builder” folder to generate feature definitions for DTP. Copy the feature definitions into the build output location’s “features” folder.
9. Zip the “eclipse” folder found in the build output location. This is your DTP SDK build file.
10. Remove one feature and three plug-ins from the build output and zip the “eclipse” folder. This is your DTP runtime build file.
11. Clean up to remove plug-in projects and build output.

You’re done!

Although there are eleven steps, the entire DTP build process is fairly fast. The problematic step is getting the code snapshot. If Eclipse.org CVS is not busy, then the DTP snapshot can be obtained in under 5 minutes. If CVS is busy, you might have to wait 30 – 40 minutes. Occasionally the script will fail, due either to network traffic (time outs) or other problems. Alternatives to get the code snapshot will be discussed below.

Once the code snapshot is obtained, the remaining DTP steps should take less than 15 minutes.

In the following sections we will detail each step listed above.

DTP Snapshot from CVS

Useful Knowledge: Familiarity with network protocols; using CVS support in Eclipse; familiarity with Ant, especially properties and using Ant from the command line.

Obviously you must first get the DTP code in order to build it. Since DTP is kept in a CVS server hosted at Eclipse.org, this means taking a snapshot over the network. The Ant script used to obtain this snapshot is found in the Build Tools distribution, “build_files\step1” and is called “cvsget.xml.” By default this script loads the DTP 1.6 stream (found in “HEAD” for each DTP module in CVS) and uses an anonymous *pserver* connection.

If Ant is in your path, at the command line located in the folder containing “cvsget.xml,” you would issue the following statement:

```
ant -f cvsget.xml
```

Ant will begin to work, and the folder will be filled with the DTP snapshot from CVS. As noted above, this script might complete within five minutes, or it could take (a lot) longer, depending on Eclipse.org bandwidth availability. Also, there might be times when the script fails: How should you get a snapshot in this situation?

If “cvsget.xml” fails, then you can still get a DTP snapshot using the Eclipse IDE: Open Eclipse, and use the Eclipse CVS tools to take a snapshot of the required DTP plug-ins (see Appendix One for the plug-in list). When using Eclipse, especially since you had problems with the *pserver* connection, you should use your committer log-in and *extssh* instead. For Eclipse.org CVS, *extssh* connections are given more bandwidth and priority, so they will rarely, if ever, fail.

So, if *extssh* works better, why not use that from the “cvsget.xml” script instead? It turns out that there is a bug in the CVS client tools that often results in a hard freeze of Ant when using *extssh*. In my experience, this has happened every time I’ve used it in the script, so I do not recommend this method. Apparently the Eclipse CVS tools use different components for *extssh*, so they do not have the same bug.

Finally, there are a number of properties for “cvsget.xml” that can be modified on the Ant command line using the “-d” syntax (see Ant documentation for details). By changing the default property values, you could load another DTP stream (such as “R1.5”), change the CVS protocol, the password, and so on. Since we do not expect usage of properties to be

required for this script, we do not detail them further here. Interested readers can check the “cvsget.xml” script directly for details.

Build in Eclipse

Suggestion: Turn “Auto-build” off in Eclipse before importing the plug-in projects. The Eclipse JDT builder sometimes gets confused if it tries to build as the plug-ins are imported.

Once you have the DTP snapshot, open Eclipse 3.3.x and import the projects into your workspace. (If you followed the instructions above, then these projects should already be in your workspace directory, so you just have to let Eclipse know about them by importing them into your Eclipse workspace.)

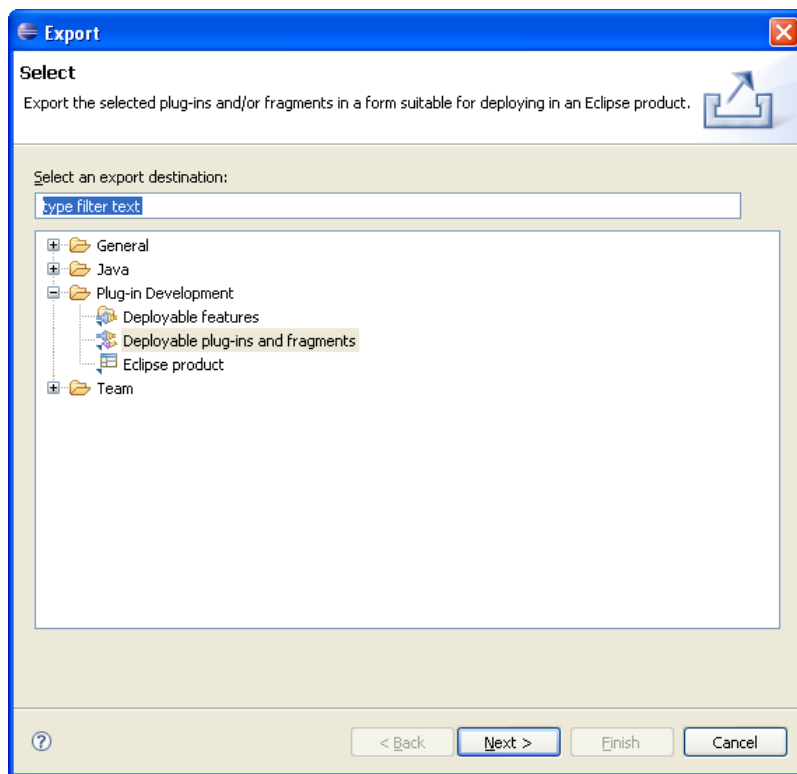
After importing the projects, do a “Build All” to make sure that you have all the necessary source code and that build errors have not been injected into the DTP CVS code line. (If build errors are injected, you need to abort the build, file a BZ report against the offending plug-in(s) and wait for the responsible committers to address the problem. In this case, an email to dtp-dev is useful to keep the community aware of why a build hasn’t completed.) Next, you have to manually (ugh, yes – *manually*) pack to specific plug-ins.

Windows has a long-standing tradition of being stupid about long path names, and XP is no better. In brief, it is a known problem that, if paths become too long, Windows will not be able to process them correctly. When unzipping in this case, you will get errors that files could not be created. When zipping, however, it is even better: *some number of files will not be added to the zip (jar) file, and no error/warning/other indication will be issued!* This means that the DTP build can produce plug-in jars that do not contain all of the necessary files, and you will not see any immediate errors. Later, when those items are not found at run time, a *ClassDefNotFound* exception or the like will be thrown. The DTP team has tried many alternatives to solve this problem (different paths structures, different Java VM versions, different registry settings, etc. etc.), but no solution to date has been identified. Apparently the only solution is to build on an OS other than Windows. We have not yet had time to port the Build Tools to another OS and verify this solution.

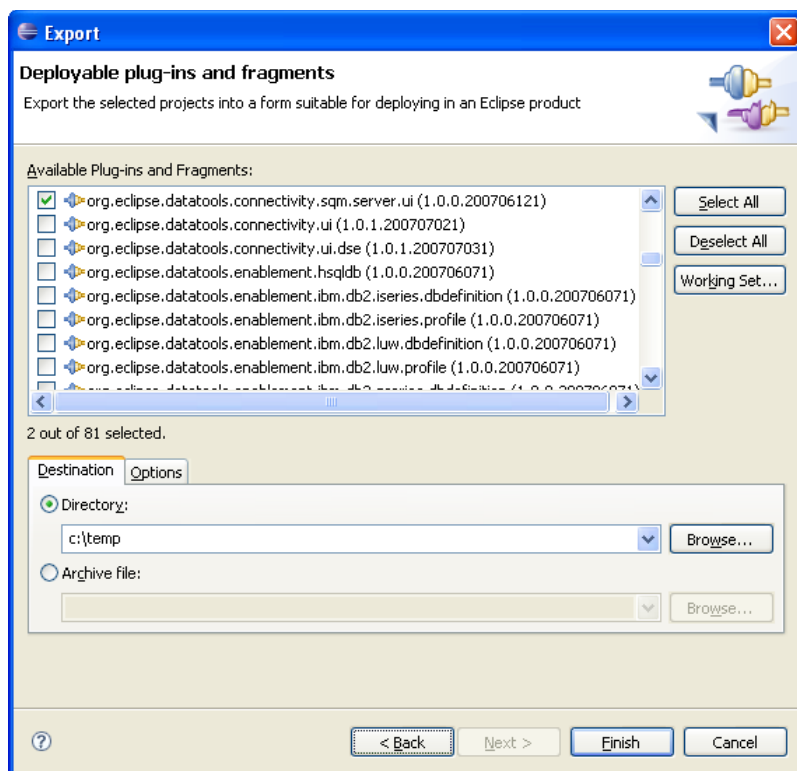
In DTP 1.6, four plug-ins that are incomplete as a result of the default build process:

- org.eclipse.datatools.connectivity.apache.derby.ui
- org.eclipse.datatools.connectivity.sqm.server.ui
- org.eclipse.datatools.enablement.ibm.db2.iseries.ui
- org.eclipse.datatools.enablement.msft.sqlserver.ui

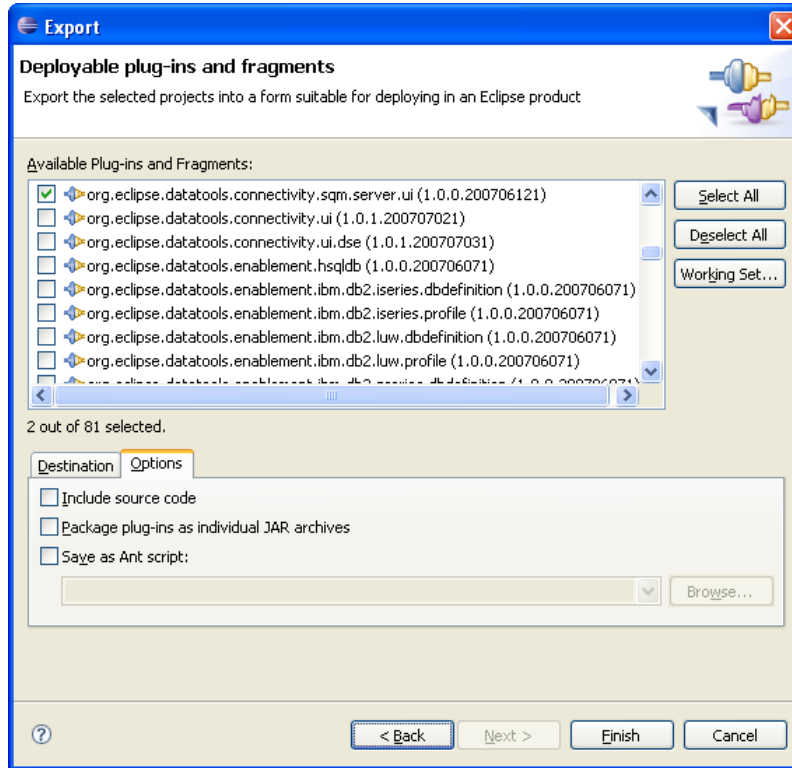
Thus, special handling is required for these plug-ins. In the Eclipse IDE, select this plug-in in the Navigator and choose “File→Export→Deployable Plug-ins and Fragments”:



Next, choose a location for the output (say, your temporary folder):



(Make sure that only the target plug-ins required is selected.) And click over to the “Options” tab:



Confirm that *none* of the three options is checked, and click on “Finish.” You now have an incomplete build for the plug-ins. Next, you’ll need to patch this plug-in to add the missing class files:

1. Navigate into the Eclipse workspace directory, to the plug-in’s “bin” folder.
2. Copy the “org” folder from that location.
3. Navigate to the exported plug-in’s folder.
4. Overwrite the existing “org” folder with a paste of the one from the workspace.
5. Select all, and zip in a “jar” archive using the pluginName_version for the name and “.jar” for the extension.

The result is that you have created plug-in jars now containing a complete set of classes for the plug-ins, and in the format expected by Eclipse. Leave it in the export location for the time being, and return to the main DTP build.

Building the DTP Plug-ins

Now you are ready to build the DTP code snapshot. From the Build Tools distribution, copy all the files found in “build_files/step2” into your snapshot folder, allowing overwrite of any existing files.

Next you will run the main part of the DTP build. The Ant script, `build.xml`, that accomplishes this is found in the “org.eclipse.datatools.build” folder. This script needs to know where the Eclipse target platform is location and what you want the script to do (that is, build all of DTP). A batch file, “build-dtp.bat” is included to encapsulate these details: you will need to tweak the Eclipse target platform path for your machine:

```
cd org.eclipse.datatools.build
ant -Declipse.home=D:\dtp_build\workfolder\eclipse_plugins_3.3.1
dtp.build
cd ..
```

The property “eclipse.home” should be set to the location of the Eclipse 3.3.x (plus EMF, GEF and Orbit jars) environment that you have set up for DTP builds. Once started, “build-dtp.bat” will work for about five minutes (YMMV of course) and does the following steps:

- Builds all DTP code snapshot plug-ins. The output is put in a “download” folder created in the snapshot directory.
- Copies static DTP content into the build output.

The DTP static content contains a number of items, and it found in the “files” folder of the DTP Built Tools “build_files/step1” directory:

- Eclipse license files
- Eclipse documentation (ISV and end user plug-ins): These are generated by a separate process on a periodic basis, and will not be documented further here.
- A feature branding plug-in for the Enablement project, since there is no “primary” carrier plug-in in that module..

If “build-dtp” completes successfully, then you will have 97 of the 98 plug-ins required for DTP 1.6 built and in the output location. If the build has errors (but presumably did not in the Eclipse IDE based build), then that singles that happy fact at least one build file is out of synch with the actual plug-in dependencies. Fixing this entails figuring out which plug-in is broken (easy – just look at the error message from Ant), opening the associated build script, comparing the class-path with the dependencies shown in the Eclipse Java Package Explorer, “Plug-in Dependencies” node, and making any necessary adjustments. A worse scenario happens when the dependencies *between* DTP plug-ins change, so the build order needs to be adjusted. This entails opening the project-specific build file (found under “org.eclipse.datatools.build” named for each project) and adjusting the build order. Fortunately, both of these situations are relatively rare in maintenance builds, since bug fixes (usually) do not inject these types of modifications. As always, feel free to abort the build and work with the committers responsible to resolve the problem.

Add Exported Plug-ins

As you will recall from above, there are a few incomplete plug-ins (missing class files) in the build output. So, copy the previously exported and zipped plug-ins into the output location's "plugins" folder, overwriting the ones already present. If you compare as each is overwritten, you will see that the manually packed ones are somewhat larger: this is correct.

Generate the Source Code Plug-in

Useful Preparation: In notepad (or some other text editor) create a DTP time stamp for the day's build. For example, a DTP 1.6 build done on 10/22/07 would be "1.6.0.200710221." You will use this value several times in the following steps.

You now have all but one DTP plug-in built, and all of the required class files should be present. The final plug-in to build is the one that wraps the DTP source code archives for the SDK distribution. In the Build Tools distribution, there is a "src_plugin" directory containing a batch file "make-src.bat" for this purpose:

```
set CLASSPATH=.
java MakeSource %1 D:\b
D:\dtp_build\build_tools\src_plugin\template\org.eclipse.datatools
.source D:\b\download\eclipse\plugins
```

You will need to tweak the paths for your install locations. The program MakeSource expects the following arguments in order:

- The version stamp. This is passed into the batch file as the first argument.
- The location of the code snapshot
- The location of the source plug-in template file. You will only have to modify the "D:\dtp_build\build_tools" part to point to the location of your Build Tools distribution.
- The output location. Here we have set it to the plugins folder in the output location of the build.

Therefore, using the example timestamp above, we would run the following on the command line:

```
make-src 1.6.0.200710221
```

The result is a plug-in added to the build output "plugins" folder containing the DTP source code. (Note that you will see a number of warnings when running this script – these are for information only, and can be ignored.)

You now have a complete set of DTP build plug-ins! The only steps remaining are to generate the feature definitions, and zip the DTP builds. But first we'll take a brief detour to check that everything (i.e. class files) is present and accounted for.

Sanity Check

As discussed above, there are situations in which archives do not always contain all of the required resources. In an attempt to check class files (at least), a small program was created to compare what was built by the Eclipse IDE in your workspace against what is present in the build output location.

A batch file, `checksanity.bat`, found in the Build Tools distribution, "sanity" folder, is provided for convenience, and will need to be tweaked for your environment's paths:

```
set CLASSPATH=.;%CLASSPATH%
java SanityCheck D:\b D:\b\download\eclipse\plugins >
sanity_out.txt
```

The program `SanityCheck` expects two arguments:

- The location of the Eclipse workspace (here, "D:\b")
- The locations of the built plug-ins

Also, we have redirected the output to a text file, `sanity_out.txt`, so the report can be more easily understood.

As an example of the errors captured by `SanityCheck`, if you were to run it immediately after a successful build ("`build-dtp.bat`"), but without patching in the two incomplete plug-ins discussed above, the following errors would be noted:

Info: Found 3508 classes...

Warning: Build has
`org.eclipse.datatools.connectivity.sqm.core/org.eclipse/datatools/connectivity/sqm/core/rte/jdbc/JDBCCatalog$1.class`
but workspace does not!

<clip>

*****Error: 13 Classes not in Build*****

`org.eclipse.datatools.connectivity.sqm.server.ui/org.eclipse/datatools/connectivity/sqm/server/internal/ui/explorer/action
s/popup/RevisedAbstractAction$NullSelectionException.class`

<clip>

The report tells us that 3508 classes have been found in the build. Lines starting with "Warning" are for information, and do not signal an actual problem. In the current DTP build stream, a number of warnings are logged by `SanityCheck`, and each of these has to

do with inner class names. Apparently there is an implementation difference in naming conventions of inner classes between the Sun Java compiler and the Eclipse JDT compiler. We have confirmed that each inner class is present in the build, so the warnings just serve to track these differences.

The line “Error: 13 Classes not in Build,” however, signals a serious problem that must be addressed. Essentially SanityCheck is reporting that some number of top-level classes are present in the Eclipse workspace but not in the output location. At present, this occurs only when the two plug-ins discussed above have not been patched into the build. *If an error is reported, the build should not be made available to others until the source of the error is determined and fixed.*

Thus, if the patched plug-ins are added to the build output location and SanityCheck is executed, then only warning items are expected in the report.

Generating the Feature Definitions

The final step before packaging the DTP distribution zips in to generate the feature definitions. The tools for doing so and feature definition templates are located in the Build Tools distribution, “site_builder” folder. There are three parts to this tool:

- The program BuildSite and its associated batch file.
- The feature templates, located in the “features” folder
- The plug-in output location, “plugins” folder.

As with the other batch files, buildsite.bat needs to be tweaked for your local environment:

```
set PATH=C:\jdk1.5.0_04\bin
set CLASSPATH=.
java BuildSite -i D:\b\download\eclipse\plugins -o D:\dtp_build\build_tools\site_builder
-v %1
```

Note that the *path* is set to a JVM 1.5 location so BuildSite will run under this VM version. This is not strictly necessary for nightly DTP builds, but is required to produce release train update sites correctly. So, if you have a 1.5 VM install location, tweak the path for that location (in case you need to produce release train update sites in the future), or simply delete this line if not. (**Note:** if you delete this line, then you will need to recompile BuildSite .java with a 1.4 VM to use it. Delete the class files in the site_builder folder, and from the command line issue “javac BuildSite .java”.)

BuildSite expects three arguments:

- -i for the location of the plug-ins built (by “build-dtp.bat”)
- -o for the output location. Note that you should set this to the location of you site_builder folder.

- -v for the version stamp, provided on the command line to the batch file.

BuildSite works by first packing all the DTP plug-ins that are not jarred into jar files (currently, the DTP build packs all but two) and copying the output to the “plugins” folder in the output location. (The update manager expects all plug-ins to be in jars, regardless of whether they are built for jar distribution or folder distribution. When DTP is installed from the Update Manager, mark-up in the feature definitions tells the update manager which plug-ins should be left as jars and which should be unpacked into folder structures in the install location.) Next BuildSite spins through the feature definitions and does the following:

- Produces a new set of features with the version stamp provided.
- Updates all the DTP plug-in versions and feature versions found.
- Updates the plug-in sizes, both packed and unpacked (if applicable).
- Updates the “unpack” attribute depending on whether the plug-in was a jar or folder structure in the build output.

Once complete the new feature definitions are found in the “features” folder. (Thus, the templates are “rolled forward” with each update – this is the intended behavior.) BuildSite should report only that it has built 25 features for 98 plug-ins and has completed. A number of error conditions can be signaled also:

- BuildSite compares the version stamp in the plug-in’s manifest.mf file with the one used on the plug-ins jar/folder. Some build scripts can cause these two values to be different and, although the Eclipse OSGI loader does not pay attention to the jar/folder’s plug-in version stamp (it cares only about the one in manifest.mf), the Update Manager will report an error. If such errors are recorded in a nightly build, then the build can still be posted for public use, but you should follow up with the plug-in owner(s) to correct the error.
- BuildSite might stop with an error saying that it couldn’t find a particular plug-in in “jarFileSizes.” This means that a DTP feature definition has a plug-in which is not in the build. Obviously this needs to be corrected before posting the build.
- BuildSite might warn that a plug-in is not included in at least one feature definition. Hence, this is an orphaned plug-ins and will not be updated correctly by the Update Manager. This should happen only when new plug-ins are added to the build but not to the feature definitions; something which is not expected in maintenance builds.

After BuildSite successfully completes, you can delete the plug-ins copied into the “plugins” folder (these are necessary for creating an update site, but we will not cover that procedure here). Finally, copy the feature definitions to the build output location into the “features” folder.

You now have a complete DTP SDK build!

Creating the DTP Download Zips

The easiest way to create the DTP download zip is to use a tool like Winzip, select the “eclipse” folder in the build output location, right click, and choose “Add to eclipse.zip.” You then rename the zip to follow the DTP SDK naming pattern:

dtp-sdk_1.6MX_N102307.zip

(Of course, you will need to adjust the time stamp for your date.) MX is the current DTP 1.6 milestone number: “M3” as of 10/22/07.

Next, in the output folder, you will need to delete four items: one feature definition and three plug-ins:

- In the “features” directory, delete the org.eclipse.datatools.sdk feature
- In the “plugins” directory, delete org.eclipse.datatools.source, org.eclipse.datatools.doc.isv, and org.eclipse.datatools.connectivity.ui.templates.

As above, zip the “eclipse” folder. You now have a DTP run time zip. Rename it to match the DTP run time zip pattern:

dtp_1.6MX_N102307.zip

(Again, adjusting the time stamp accordingly.)

While the exact size may vary somewhat, DTP 1.6 builds are around 23.5meg for the SDK and 9.8meg for the run time. If the size is dramatically different on either zip, then you should take a moment to determine if

- You are using a different zip program or different level of compression
- If smaller, have you forgotten to add something to the build (SanityCheck and BuildSite will usually catch this, but....)

Note that you can always compare existing DTP builds to the new one using standard directory compare tools. (In general, any size difference of less than 1meg is nothing to worry about – apparently zip is somewhat heuristic, and varies slightly day-to-day for exact content.)

With these two zips in hand, the DTP build is complete. Next, we clean up.

Post Build Clean Up

After the build is complete, you can delete all of the plug-ins using the Eclipse IDE, and then select everything except for the “.metadata” This will give you a clean environment in which to build again in the future.

Appendix One: DTP Code Snapshot for 1.6

Below is a list of plug-ins in the DTP 1.6 code snapshot to be obtained from CVS, grouped by CVS module (“DTP project”). Note that this does not include static content added during the build (see Appendix Two for the list of DTP build plug-ins). For DTP 1.6, all of these are found in the modules’ “HEAD.”

org.eclipse.datatools.build module

org.eclipse.datatools.help
org.eclipse.datatools.oda.cshelp

org.eclipse.datatools.connectivity module

org.eclipse.datatools.connectivity.apache.derby.dbdefinition
org.eclipse.datatools.connectivity.apache.derby.ui
org.eclipse.datatools.connectivity.apache.derby
org.eclipse.datatools.connectivity.db.generic.ui
org.eclipse.datatools.connectivity.db.generic
org.eclipse.datatools.connectivity.dbdefinition.genericJDBC
org.eclipse.datatools.connectivity.oda.consumer
org.eclipse.datatools.connectivity.oda.design.ui
org.eclipse.datatools.connectivity.oda.design
org.eclipse.datatools.connectivity.oda.flatfile.ui
org.eclipse.datatools.connectivity.oda.flatfile
org.eclipse.datatools.connectivity.oda.profile
org.eclipse.datatools.connectivity.oda.template.ui
org.eclipse.datatools.connectivity.oda
org.eclipse.datatools.connectivity.sqm.core.ui
org.eclipse.datatools.connectivity.sqm.core
org.eclipse.datatools.connectivity.sqm.server.ui
org.eclipse.datatools.connectivity.ui.dse
org.eclipse.datatools.connectivity.ui.templates
org.eclipse.datatools.connectivity.ui
org.eclipse.datatools.connectivity

org.eclipse.datatools.enablement module

org.eclipse.datatools.enablement.hsqldb
org.eclipse.datatools.enablement.ibm.db2.iseries.dbdefinition
org.eclipse.datatools.enablement.ibm.db2.iseries.ui
org.eclipse.datatools.enablement.ibm.db2.iseries
org.eclipse.datatools.enablement.ibm.db2.luw.dbdefinition
org.eclipse.datatools.enablement.ibm.db2.luw.ui
org.eclipse.datatools.enablement.ibm.db2.luw
org.eclipse.datatools.enablement.ibm.db2.zseries.dbdefinition
org.eclipse.datatools.enablement.ibm.db2.zseries.ui
org.eclipse.datatools.enablement.ibm.db2.zseries
org.eclipse.datatools.enablement.ibm.informix.dbdefinition
org.eclipse.datatools.enablement.ibm.informix.ui
org.eclipse.datatools.enablement.ibm.informix
org.eclipse.datatools.enablement.ibm.ui
org.eclipse.datatools.enablement.jdt.classpath
org.eclipse.datatools.enablement.msft.sqlserver.dbdefinition
org.eclipse.datatools.enablement.msft.sqlserver.ui
org.eclipse.datatools.enablement.msft.sqlserver
org.eclipse.datatools.enablement.mysql.dbdefinition
org.eclipse.datatools.enablement.mysql.ui
org.eclipse.datatools.enablement.mysql
org.eclipse.datatools.enablement.oda.ws.ui
org.eclipse.datatools.enablement.oda.ws
org.eclipse.datatools.enablement.oda.xml.ui
org.eclipse.datatools.enablement.oda.xml
org.eclipse.datatools.enablement.oracle.dbdefinition
org.eclipse.datatools.enablement.oracle.ui
org.eclipse.datatools.enablement.oracle
org.eclipse.datatools.enablement.postgresql.dbdefinition
org.eclipse.datatools.enablement.postgresql.ui
org.eclipse.datatools.enablement.sap.maxdb
org.eclipse.datatools.enablement.sap.maxdb.dbdefinition
org.eclipse.datatools.enablement.sap,maxdb.ui
org.eclipse.datatools.enablement.postgresql
org.eclipse.datatools.enablement.sybase.asa.models
org.eclipse.datatools.enablement.sybase.asa
org.eclipse.datatools.enablement.sybase.ase.dbdefinition
org.eclipse.datatools.enablement.sybase.ase.profile
org.eclipse.datatools.enablement.sybase

org.eclipse.datatools.modelbase module

org.eclipse.datatools.modelbase.dbdefinition
org.eclipse.datatools.modelbase.derby

1/30/2008

org.eclipse.datatools.modelbase.sql.edit
org.eclipse.datatools.modelbase.sql.query.edit
org.eclipse.datatools.modelbase.sql.query
org.eclipse.datatools.modelbase.sql.xml.query
org.eclipse.datatools.modelbase.sql

org.eclipse.datatools.sqltools module

org.eclipse.datatools.sqltools.common.ui
org.eclipse.datatools.sqltools.data.core
org.eclipse.datatools.sqltools.data.ui
org.eclipse.datatools.sqltools.db.derby
org.eclipse.datatools.sqltools.db.generic
org.eclipse.datatools.sqltools.ddlgen.ui
org.eclipse.datatools.sqltools.debugger.core
org.eclipse.datatools.sqltools.editor.core
org.eclipse.datatools.sqltools.parsers.sql.lexer
org.eclipse.datatools.sqltools.parsers.sql.query
org.eclipse.datatools.sqltools.parsers.sql.xml.query
org.eclipse.datatools.sqltools.parsers.sql
org.eclipse.datatools.sqltools.plan
org.eclipse.datatools.sqltools.result
org.eclipse.datatools.sqltools.routineeditor
org.eclipse.datatools.sqltools.sqlbuilder
org.eclipse.datatools.sqltools.sqleditor
org.eclipse.datatools.sqltools.sqlscrapbook
org.eclipse.datatools.sqltools.sql
org.eclipse.datatools.sqltools.tabledataeditor
org.eclipse.datatools.sqltools.tablewizard

Appendix Two: DTP Build Plug-ins

The plug-ins expected in a DTP 1.6 build are listed below. These include both plug-ins from CVS and those added as static or generated content (in italics):

javax.wsdl
net.sourceforge.lpg.lpgjavaruntime
org.apache.xerces
org.apache.xml.resolver
org.eclipse.datatools.connectivity.apache.derby.dbdefinition
org.eclipse.datatools.connectivity.apache.derby.ui
org.eclipse.datatools.connectivity.apache.derby
org.eclipse.datatools.connectivity.db.generic.ui
org.eclipse.datatools.connectivity.db.generic
org.eclipse.datatools.connectivity.dbdefinition.genericJDBC
org.eclipse.datatools.connectivity.oda.consumer

org.eclipse.datatools.connectivity.oda.design.ui
org.eclipse.datatools.connectivity.oda.design
org.eclipse.datatools.connectivity.oda.flatfile.ui
org.eclipse.datatools.connectivity.oda.flatfile
org.eclipse.datatools.connectivity.oda.profile
org.eclipse.datatools.connectivity.oda.template.ui
org.eclipse.datatools.connectivity.oda
org.eclipse.datatools.connectivity.sqm.core.ui
org.eclipse.datatools.connectivity.sqm.core
org.eclipse.datatools.connectivity.sqm.server.ui
org.eclipse.datatools.connectivity.ui.dse
org.eclipse.datatools.connectivity.ui.templates
org.eclipse.datatools.connectivity.ui
org.eclipse.datatools.connectivity
org.eclipse.datatools.doc.isv
org.eclipse.datatools.doc.user
org.eclipse.datatools.enablement.finfo
org.eclipse.datatools.enablement.hsqldb
org.eclipse.datatools.enablement.ibm.db2.iseries.dbdefinition
org.eclipse.datatools.enablement.ibm.db2.iseries.ui
org.eclipse.datatools.enablement.ibm.db2.iseries
org.eclipse.datatools.enablement.ibm.db2.luw.dbdefinition
org.eclipse.datatools.enablement.ibm.db2.luw.ui
org.eclipse.datatools.enablement.ibm.db2.luw
org.eclipse.datatools.enablement.ibm.db2.zseries.dbdefinition
org.eclipse.datatools.enablement.ibm.db2.zseries.ui
org.eclipse.datatools.enablement.ibm.db2.zseries
org.eclipse.datatools.enablement.ibm.informix.dbdefinition
org.eclipse.datatools.enablement.ibm.informix.ui
org.eclipse.datatools.enablement.ibm.informix
org.eclipse.datatools.enablement.ibm.ui
org.eclipse.datatools.enablement.jdt.classpath
org.eclipse.datatools.enablement.msft.sqlserver.dbdefinition
org.eclipse.datatools.enablement.msft.sqlserver.ui
org.eclipse.datatools.enablement.msft.sqlserver
org.eclipse.datatools.enablement.mysql.dbdefinition
org.eclipse.datatools.enablement.mysql.ui
org.eclipse.datatools.enablement.mysql
org.eclipse.datatools.enablement.oda.ws.ui
org.eclipse.datatools.enablement.oda.ws
org.eclipse.datatools.enablement.oda.xml.ui
org.eclipse.datatools.enablement.oda.xml
org.eclipse.datatools.enablement.oracle.dbdefinition
org.eclipse.datatools.enablement.oracle.ui
org.eclipse.datatools.enablement.oracle
org.eclipse.datatools.enablement.postgresql.dbdefinition

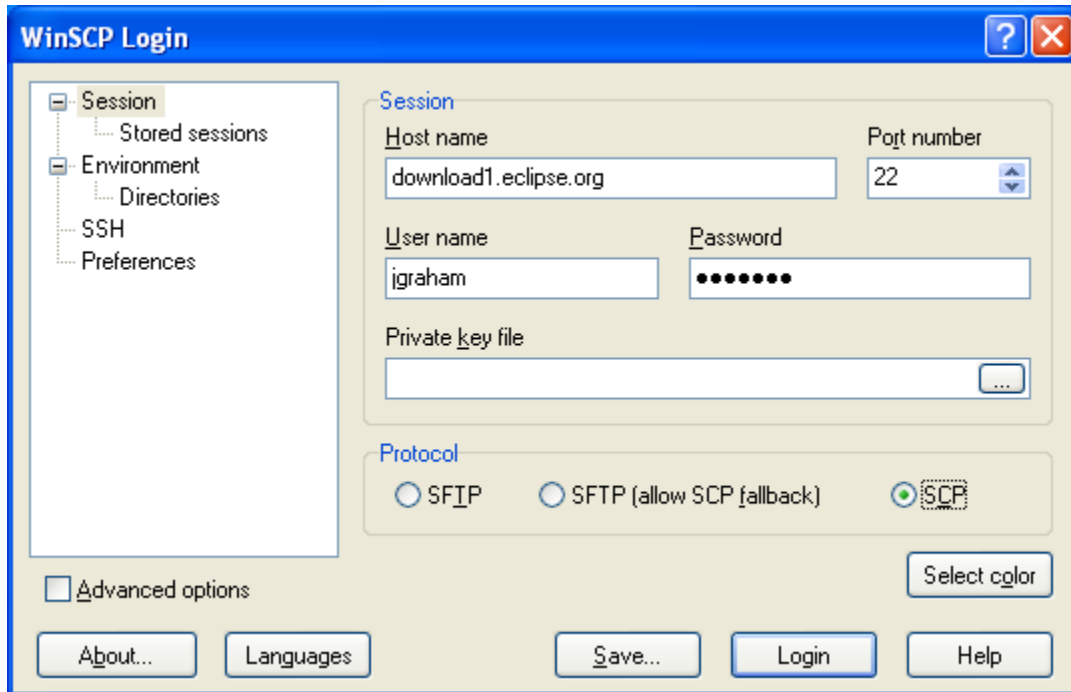
org.eclipse.datatools.enablement.postgresql.ui
org.eclipse.datatools.enablement.postgresql
org.eclipse.datatools.enablement.sap.maxdv.dbdefinition
org.eclipse.datatools.enablement.sap.maxdb.ui
org.eclipse.datatools.enablement.sap.maxdb
org.eclipse.datatools.enablement.sybase.asa.models
org.eclipse.datatools.enablement.sybase.asa
org.eclipse.datatools.enablement.sybase.ase.dbdefinition
org.eclipse.datatools.enablement.sybase.ase.profile
org.eclipse.datatools.enablement.sybase
org.eclipse.datatools.help
org.eclipse.datatools.intro
org.eclipse.datatools.modelbase.dbdefinition
org.eclipse.datatools.modelbase.derby
org.eclipse.datatools.modelbase.sql.edit
org.eclipse.datatools.modelbase.sql.query.edit
org.eclipse.datatools.modelbase.sql.query
org.eclipse.datatools.modelbase.sql.xml.query
org.eclipse.datatools.modelbase.sql
org.eclipse.datatools.oda.cshelp
org.eclipse.datatools.source
org.eclipse.datatools.sqltools.common.ui
org.eclipse.datatools.sqltools.data.core
org.eclipse.datatools.sqltools.data.ui
org.eclipse.datatools.sqltools.db.derby
org.eclipse.datatools.sqltools.db.generic
org.eclipse.datatools.sqltools.ddlgen.ui
org.eclipse.datatools.sqltools.debugger.core
org.eclipse.datatools.sqltools.editor.core
org.eclipse.datatools.sqltools.parsers.sql.lexer
org.eclipse.datatools.sqltools.parsers.sql.query
org.eclipse.datatools.sqltools.parsers.sql.xml.query
org.eclipse.datatools.sqltools.parsers.sql
org.eclipse.datatools.sqltools.plan
org.eclipse.datatools.sqltools.result
org.eclipse.datatools.sqltools.routineeditor
org.eclipse.datatools.sqltools.sqlbuilder
org.eclipse.datatools.sqltools.sqleditor
org.eclipse.datatools.sqltools.sqlscrapbook
org.eclipse.datatools.sqltools.sql
org.eclipse.datatools.sqltools.tabledataeditor
org.eclipse.datatools.sqltools.tablewizard

Appendix Three: Uploading the DTP Builds

This section details the steps required for making DTP 1.6 builds available on the Eclipse.org download site. It is assumed that the reader has a current Eclipse.org committer account.

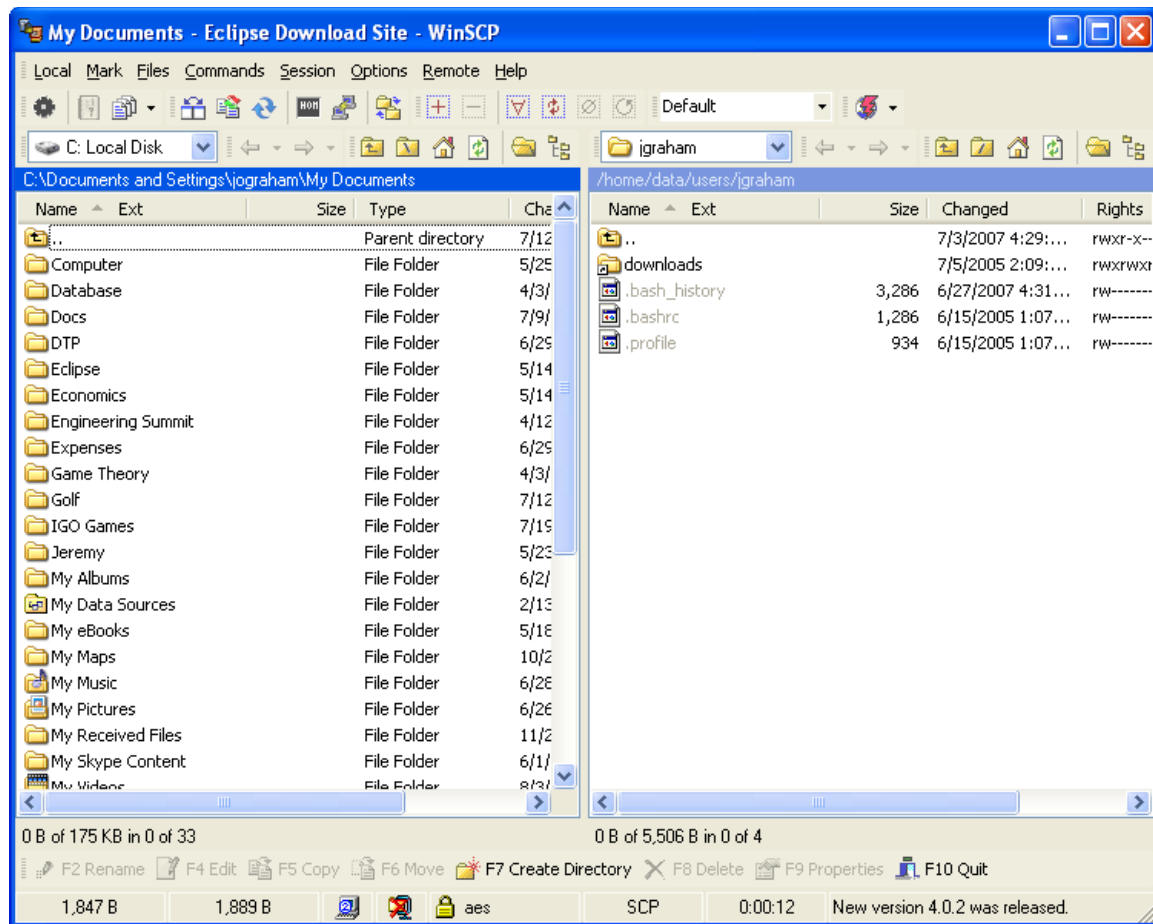
To send files to the Eclipse.org download server, you will need to have an SCP client program. I suggest WinSCP (<http://winscp.net/eng/index.php>), and the rest of this section will be written assuming usage of that tool.

The first time you start WinSCP after installing it, a connection configuration dialog will appear:



I've added the host name "download1.eclipse.org," left the port at the default 22 value, entered my committer id and password, and selected "SCP" for the protocol. Rather than simply clicking on "Login," first click "Save" so you can reuse these values later.

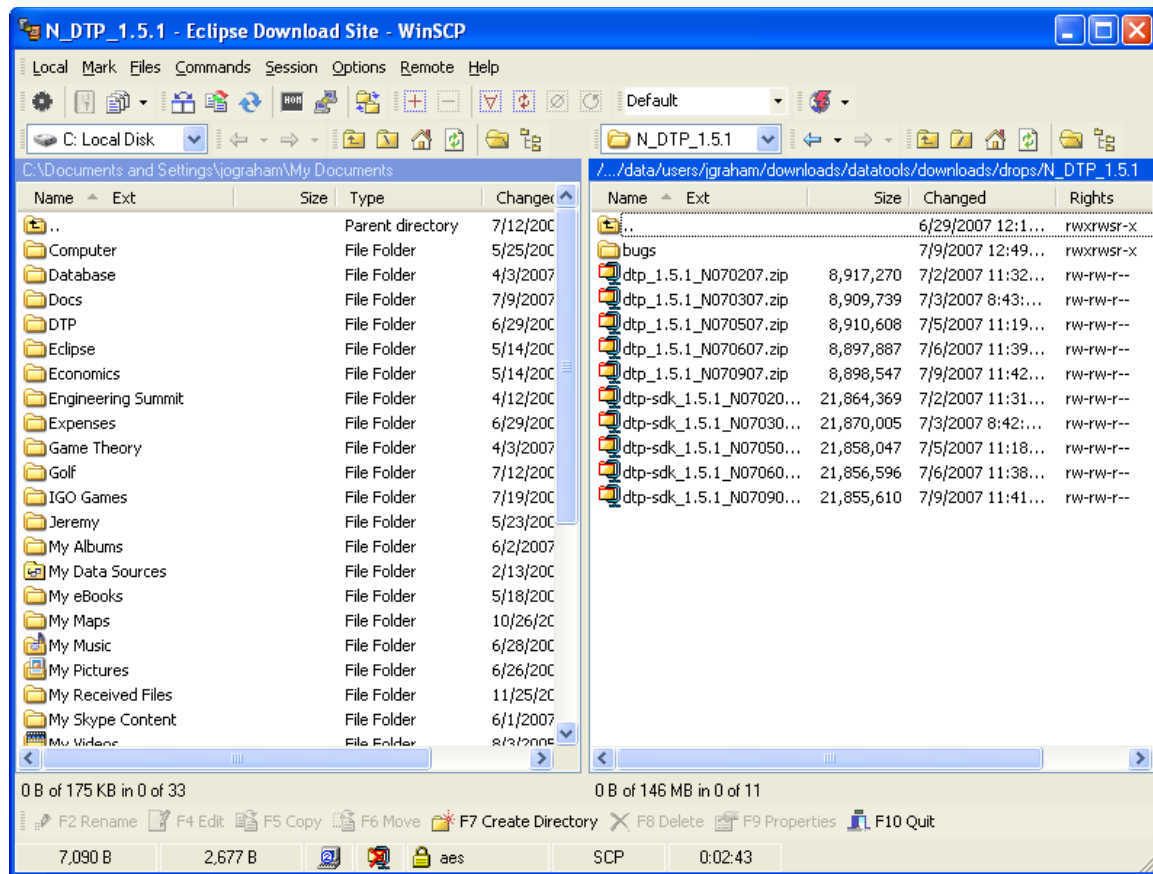
On the first connection to the Eclipse download server, WinSCP will complain that the site key is not registered. Accept the site key and let WinSCP make the connection:



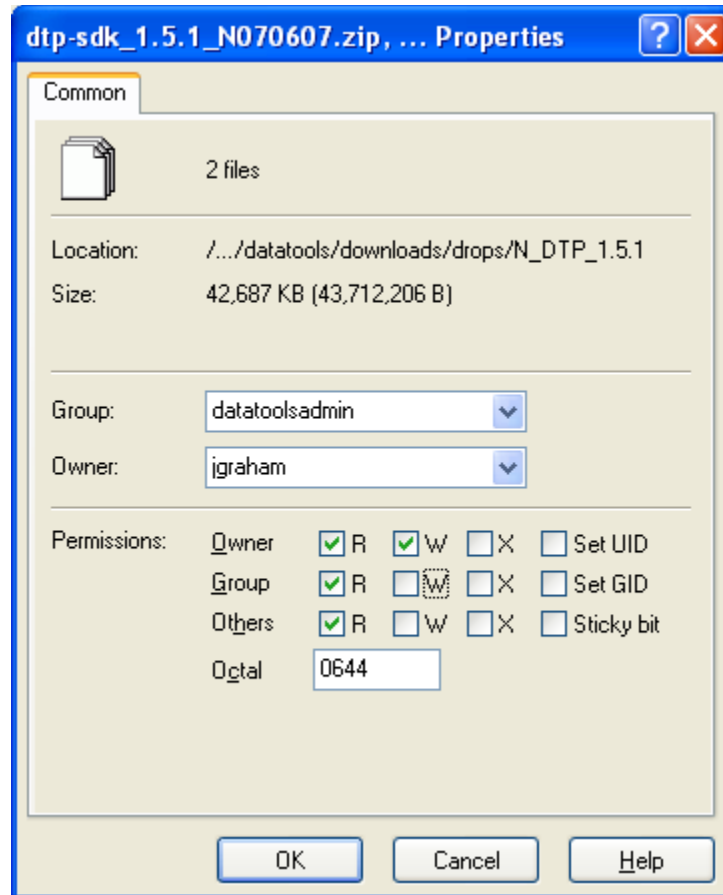
The left-hand pane is your local machine, and the right-hand one is the Eclipse download server. First, navigate to the Data Tools download area by clicking through the following folder sequence:

downloads→datatools→downloads→drops→N_DTP_1.6

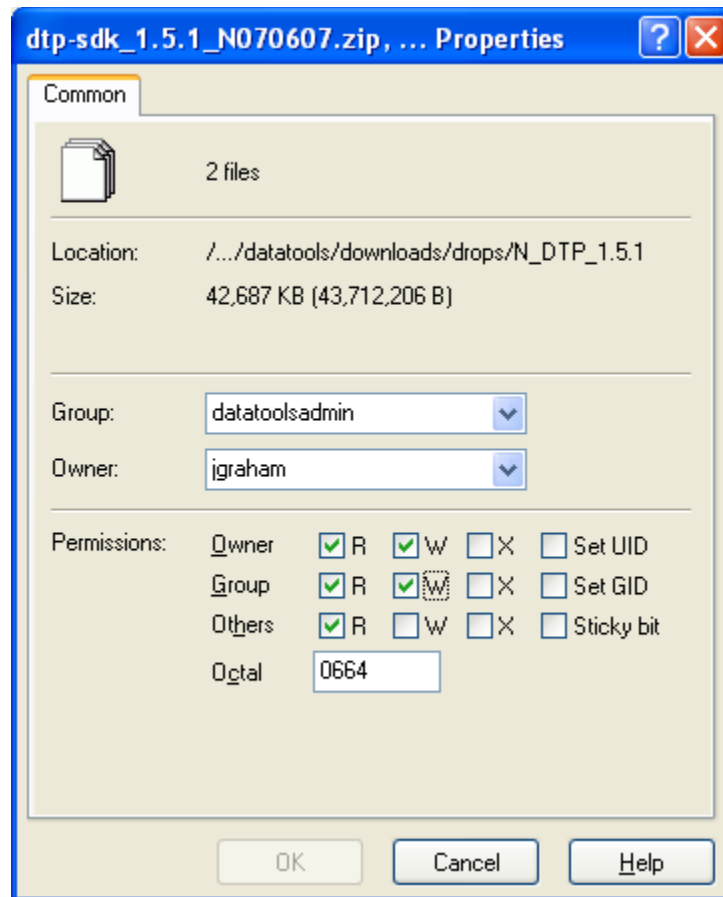
You are now at the DTP 1.6 download folder:



Next, you would navigate in the left-hand pane to the directory containing the DTP build files you want to upload. You can simply select and drag/drop these files into the right-hand pane to start the upload. Once the upload is complete, you need to set the correct permissions on the new files. Select them in the right-hand pane, right click, select "Properties":



Initially the Group→W (write) flag will not be set. Click to set this flag:



And click on “OK.” This is necessary in order for the Eclipse download infrastructure to more easily identify these files as candidates for downloads. You can then close WinSCP.

Typically it takes between five to thirty minutes (depending on Eclipse.org load) for the newly uploaded files to be available for download. Note that we are using a folder for DTP 1.6 nightly builds that begins with “N.” These folders *are not* mirrored (that is, the only download server hosting them is the main Eclipse download site) to reduce load/replication traffic on mirrors.

The download URL follows the patterns below:

http://www.eclipse.org/downloads/download.php?file=/datatools/downloads/drops/N_DTP_1.6/dtp-sdk_1.6M3_N102207.zip

http://www.eclipse.org/downloads/download.php?file=/datatools/downloads/drops/N_DTP_1.6/dtp_1.6M3_N102207.zip

Note that there is *no line break* in these URL and that the version stamp needs to be tweaked for the new build files uploaded. Once the new build zips are accessible on the Eclipse download site, links like those above can be sent to anyone wishing to download the new zips.