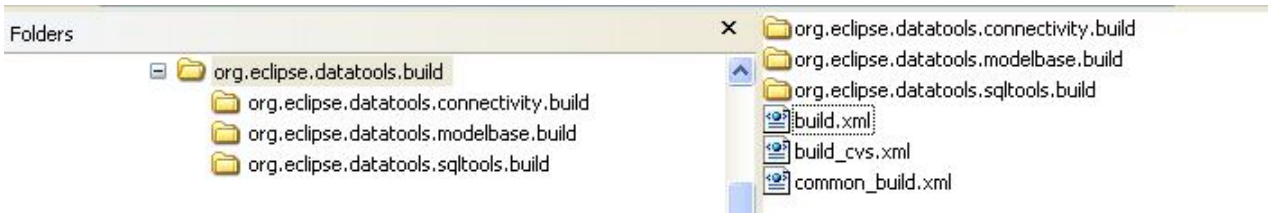
The overall DTP build files will be kept in a separate CVS module *org.eclipse.datatools.build*. The contents of this module are shown below:



There are three files at the top-level:

1. *build.xml*: The overall Ant build file for DTP, used to produce download plug-in sets.
2. *build_cvs.xml*: The Ant script responsible for checking out a copy of the DTP plug-ins to be built from Eclipse CVS. Used by the top-level build script described in (1).
3. *common_build.xml*: A set of Ant targets common to all DTP plug-in build files.

Also included are folders for each DTP project, each containing an Ant *build.xml* file responsible for building all the plug-ins contained in that DTP project. These project-level build scripts are used by the top-level build file. Thus, the build sequence for DTP is:

1. Check out a copy of the *org.eclipse.datatools.build* module
2. Execute *build.xml* specifying the location of the target Eclipse platform ("eclipse.home" property)
    a. Call the "checkout.dtp" target, found in *build_cvs.xml*, to obtain the plug-in set. These are copied into a folder one level up from *org.eclipse.datatools.build*.[1]
    b. Call the default target for the build script found in the *org.eclipse.datatools.modelbase.build* folder.
        i. Set properties indicating a top-level DTP build in progress and the target "download" folder
        ii. Execute the build script for each Model Base plug-in using the "download.plugin" target.
    c. Call the default target for the build script for in the *org.eclipse.datatools.connectivity.build* folder.
        i. Set properties indicating a top-level DTP build in progress and the target "download" folder
        ii. Execute the build script for each Connectivity plug-in using the "download.plugin" target.
    d. Call the default target for the build script for in the *org.eclipse.datatools.sqltools.build* folder.

[1] In the current build file, this target is not called by default – it has to be used separately. This is because the updated build scripts for each plug-in have to copied over after check out. CVS checkout will be linked to the main build target when each DTP plug-in has an updated build file present in CVS.

i. Set properties indicating a top-level DTP build in progress and the target "download" folder
ii. Execute the build script for each SQL Development Tools plug-in using the "download.plugin" target.

The result is a collection of plug-ins in the "download" folder, representing a complete DTP build. Each plug-in includes:

1. Jar files for the built source
2. A zip file containing the plug-in source code
3. All of the other support files, such as the manifest, *plugin.xml*, and so on. These files are specified in the "bin includes" part of each plug-ins' build script.
4. JavaDoc for the plug-in, if applicable, in the "doc" folder for the plug-in.

As described in previous discussions about the DTP build system, there are several main goals:

1. Ease of use: Anyone with access to Eclipse CVS should be able to obtain the top-level build files and run them with minimal configuration. The result should be a build equivalent to that produced by the official DTP build.
2. Multiple target environments: The build scripts for DTP plug-ins should be executable in standard Eclipse IDE environments, as well as in the command-line DTP build. This is accomplished by keeping changes to the PDE-generated build file minimal, and making inclusion of top-level requirements, such as *common_build.xml*, optional.
3. Allow for build options: Users are still able to call the standard (generated) build targets to get, for example, only the Jar file, or compile sources. Also, the top-level build can be executed with or without JavaDoc generation.
4. Simple to maintain: The build files included with each plug-in must now be regarded as project artifacts that have to be maintained, just as source code or the plug-in manifest file. We have tried to make the maintenance burden as small as possible, but it is not zero (see below).

This build system works today (as of the 1/6/06 iteration build), but will require attention to maintain going forward. As stated in goal (4), we have tried to limit the maintenance burden imposed on DTP plug-in developers. At a minimum, a basic understanding of Ant build scripts will be required. The following items explain the points that must be maintained as plug-ins evolve.

1. The *build.properties* file for a plug-in must contain a set of standard properties. When the build files are checked into CVS, each plug-ins *build.properties* file will be updated to contain these. You will notice that there is redundancy in property values as a result of adding these common values. Individual plug-in owners are free to leave these values as-is, or modify their *build.properties* and *build.xml* files to eliminate them (of course, keeping the standard properties

required by the DTP top-level build). Note that the *build.properties* file can still be modified in the Eclipse standard editor.

2. As part of the standard properties added to *build.properties*, the plug-in version is given in "plugin.version". This needs to be synchronized with the value contained in the plug-in manifest. Unless otherwise stated, all DTP plug-ins should carry a version of "0.7.x" for the first (March) release. As part of the Eclipse release train ("Callisto"), we will need to update the file version number each time we change a specific plug-in, to insure that that Callisto update manager makes the correct versions of DTP plug-ins available for download. Note that this property is used throughout the build files to specify the version number: No hard-coded version numbers should appear in build files.
3. Another standard property "module.name" is used to identify the plug-in during the DTP build process. This property has been used in the build scripts as necessary.
4. Each build script contains the "eclipse.home" property, set to current folder by default. When running build files inside the Eclipse IDE, this property will need to be set in the Ant launcher. This value is also set in the top-level DTP build, allowing any Eclipse platform version to be easily built against.
5. Each build script contains a set of parameters used by the common build file to generate JavaDoc. You can change the browser title value, the main JavaDoc page tile, the packages exposed for JavaDoc, and the copyright footer. For each build script, as best guess has been made, and we should try to follow a standard in DTP for these values. Finally, an additional property "add.javadoc" can be set to "false" to skip JavaDoc generation. This is used for DTP plug-ins where JavaDoc is not required, and also can be used by any build process as a command-line parameter to skip all JavaDoc generation.
6. Several values for the Java compiler are set: All DTP build scripts should fail on Java compiler errors (this allows us to stop the build and determine the source of build errors much more quickly than having a cascade of errors throughout an entire build). The source code and byte-code compliance levels are set by parameters in the *build.properties* files. The base values for these in DTP is "1.4" for both. (This represents the minimal compiler/JVM.)
7. The PDE-generated "gather.bin.parts" is extended slightly to include a zip of the plug-in source code.

Each of the above seven points is either static or easily updated (e.g. version number). There are two key areas in the build scripts that likely will change frequently over plug-in development. These are:

1. In the "gather.bin.parts" target, there is a Ant "copy" operation that copies all of the resources specified in the *build.properties* file to the output folder. This is correct when the script is generated by PDE, but can fall out of synch as the plug-in evolves. Thus, this value needs to be manually updated whenever there are changes to the binary resource set in *build.properties*.
2. The classpath for building the plug-in is derived from the manifest by the PDE during script generation. These entries have been moved into the

"path_bootclasspath" value (Ant script "path" element), and need to be updated as the plug-in's classpath changes, or else the plug-in build will fail with compile errors.

The two items above, especially the second, represent the heaviest burdens for plug-in developers in maintaining the DTP build files. (Fortunately, since DTP is designed to be easy to build, whether a build script works or not can be tested quickly. ☺)

Also note that custom Ant tasks could be written to handle each of the two points immediately above. That is, a custom Ant task could use the manifest, plug-in and *build.properties* files included with the plug-in to generate the bin-includes and the current classpath value. If we had such tasks, they could be wrapped in a Jar and included in the top-level build.