Migrating JDeveloper Projects to 10.1.3

An Oracle White Paper
February 2005

ORACLE

Migrating JDeveloper Projects to 10.1.3

INTRODUCTION

Starting with the developer preview release of JDeveloper 10.1.3, the way the product handles projects has changed to address issues with supporting teams of developers. With JDeveloper 10.1.3, the project file no longer maintains a list of every single file contained within the project, but rather uses a set of paths combined with filters to construct the list of files dynamically. This new mechanism for determining the project contents reduces the number of updates to the project file during development, which in turn reduces the chance of merge conflicts among teams of developers. JDeveloper 10.1.3 provides an automatic migration of your project to the new format based on the list of files contained in the 10.1.2 project file. In most cases, the fact that JDeveloper 10.1.3 is using a new mechanism for storing the contents of the project will be completely transparent to you. Some pre-migration steps will also be examined to simplify the migration process for complex projects. If you run into any issues post-migration, it may be helpful to understand how projects are now stored so that you may make manual adjustments to your project or the directory layout of your source.

JDEVELOPER PROJECTS 10.1.2 AND EARLIER

In this context, a "10.1.2 project" is a project created with a version of JDeveloper older than 10.1.3.

In JDeveloper projects (10.1.2 and earlier), the project file (.jpr file) typically contains a list of all of the files that comprise the project. This list is sometimes referred to as the project's "list of children." JDeveloper also used to support a feature known as dynamic project, controlled by a setting labeled "Scan Source Paths to Determine Project Contents". JDeveloper 10.1.2 dynamic projects did not include any filtering support, and not all features, namely those with composite nodes such as EJB and BC4J objects, worked well in this environment.

Advantages

The main advantage of the 10.1.2 approach is that you can have one source directory shared among various projects, picking and choosing at the file level, which files belong to each project. Another benefit of this approach is that classes that are not in a compilable state can exist in your source path and they will not be compiled nor or even known about by JDeveloper. However, this project layout is not optimal, and it is recommended to use separate source directories for each project.

Disadvantages

The 10.1.2 approach also has several disadvantages. The main problem is that every time a file is added or removed from the project file, the project file is updated. When many developers are working on a common project, it is easy for merge conflicts to be introduced if files are added or removed that might be located at the same place in the project file. Files created outside of JDeveloper must also be added to the project, as the IDE only knows about the files explicitly listed in the project file. Finally, the project file can become very large as your project grows, increasing the chance of conflicts with many updates occurring simultaneously from multiple developers.

Example of a 10.1.2 Project

The verbose structure in the 10.1.2 JDeveloper projects increases the chance of a merge conflict when multiple developers update the project file at the same time.

```
<listOfChildren class="oracle.ide.model.DataList">
   <Item class="oracle.ide.model.Reference">
      <URL path="src/mypackage/Application1.java"/>
      <nodeClass>oracle.jdeveloper.model.JavaSourceNode</nodeClass>
   </Item>
   <Item class="oracle.ide.model.Reference">
      <URL path="src/mypackage/closefile.gif"/>
      <nodeClass>oracle.jdeveloper.model.GifImageNode</nodeClass>
   </Item>
   <Item class="oracle.ide.model.Reference">
      <URL path="src/mypackage/Frame1_AboutBoxPanel1.java"/>
      <nodeClass>oracle.jdeveloper.model.JavaSourceNode</nodeClass>
   </Item>
   <Item class="oracle.ide.model.Reference">
      <URL path="src/mypackage/Frame1.java"/>
      <nodeClass>oracle.jdeveloper.model.JavaSourceNode</nodeClass>
   </Item>
   <Item class="oracle.ide.model.Reference">
      <URL path="src/mypackage/help.gif"/>
      <nodeClass>oracle.jdeveloper.model.GifImageNode</nodeClass>
   </Item>
   <Item class="oracle.ide.model.Reference">
      <URL path="src/mypackage/openfile.gif"/>
      <nodeClass>oracle.jdeveloper.model.GifImageNode</nodeClass>
   </Item>
</listOfChildren>
```

Looking at the source of this 10.1.2 project file, you may notice that each individual file has a separate entry for that file. Besides being extremely verbose, this format is quite difficult to edit by hand, when required to manually resolve a merge conflict.

JDEVELOPER 10.1.3 PROJECTS

Starting with JDeveloper 10.1.3, the project file stores a list of paths called content sets. In addition to listing a set of directories that make up the content set, each directory included also contains a list of subdirectories beneath it to exclude. By manipulating the content sets you create a dynamic list of files contained by your JDeveloper project.

Advantages

By not keeping a manual list of files in the project file, once you create a file in a directory contained within one of the project's content sets, it is automatically included in the JDeveloper project. The file system containing the source files becomes the source of truth of the contents of your project. When working in a team environment, the project file is no longer frequently updated. These frequent updates are a source of potential "merge conflicts" that often require manual intervention to bring into a consistent state after simultaneous updates.

Disadvantages

If you only want to include a few files from a directory containing many files, this approach does not work as well as individually listing out the files. Later on we will examine some ways to handle this specific case.

Example of a 10.1.3 Project

```
<hash n="oracle.jdeveloper.model.PathsConfiguration">
  <hash n="javaContentSet">
    <list n="constituent-sets"/>
    <list n="pattern-filters">
      <string v="+**/"/>
    </list>
    <list n="url-path">
      <url path="src/"/>
    </list>
  </hash>
</hash>
```

In JDeveloper 10.1.3, much less information is required in the project file, and the file system becomes the "source of truth" for the list of files contained in the project.

NEW JDEVELOPER PROJECT CONCEPTS

Many of the concepts of how projects are structured are new to JDeveloper 10.1.3. Here is a list of the new concepts to be aware of with your projects.

Project Content

JDeveloper stores the various paths that comprise the project file in a set of content sets. JDeveloper contains several different content sets such as Resources and Java. Depending on the configuration of your JDeveloper environment and which extensions are currently installed, this list may include other content sets as well. Although there may be several different entries in the project properties under the Project Content node, the default content set is the Java content set. A content set is comprised of the following:

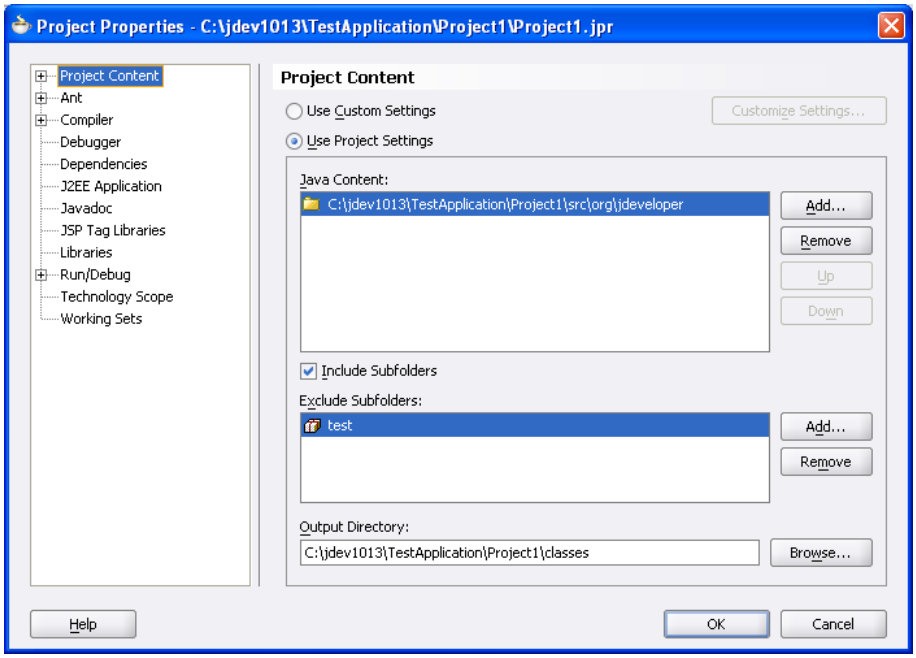
- A list of directories that contain files.
- Flag to indicate if the directory should be recursively scanned for files.
- If the directory should be recursively scanned for files, then a list of subdirectories to exclude from the search may be optionally specified.

Here is the list of files contained on our file system:

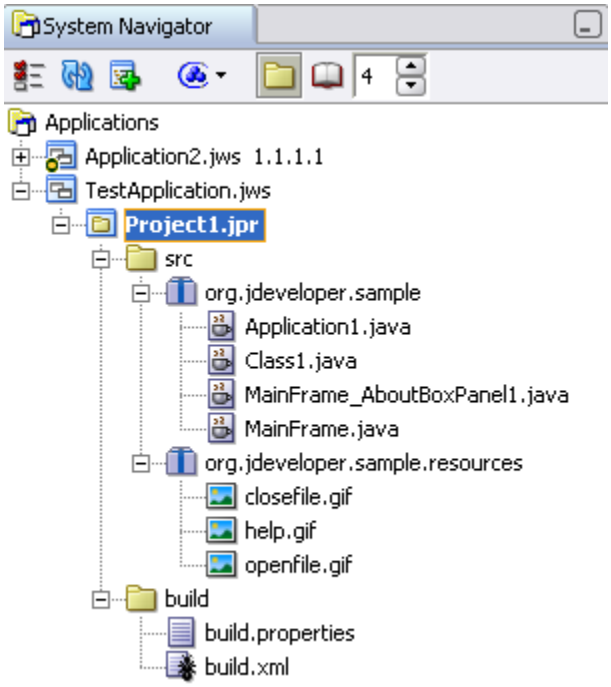
```
build\build.properties
build\build.xml
src\mypackage\OldClass1.java
src\org\jdeveloper\sample\Application1.java
src\org\jdeveloper\sample\Class1.java
src\org\jdeveloper\sample\MainFrame.java
src\org\jdeveloper\sample\MainFrame_AboutBoxPanel1.java
src\org\jdeveloper\sample\resources
src\org\jdeveloper\sample\resources\closefile.gif
src\org\jdeveloper\sample\resources\help.gif
src\org\jdeveloper\sample\resources\openfile.gif
src\org\jdeveloper\test\SimpleTest.java
```

Here is the Java content set in our Project:

The directories that comprise the Java Content Set will become the source path entries passed to the Java compiler.



Here is the set of files included in the JDeveloper Project:



We can tell from the settings of the Java content set that JDeveloper will process the following:

- Look for Java source files recursively starting at directory src\org\jdeveloper
- Ignore anything under directory src\org\jdeveloper\test

This means that the following files will not be contained in the JDeveloper project:

- src\org\jdeveloper\test\SimpleTest.java
- src\mypackage\OldClass1.java

Implicit Source Path

JDeveloper automatically derives the source path based on the directories located in the Java Content Set.

Most Java tools require you to know the source path, which are one or more root directories that comprise your project. In our example, the source path is "src".

JDeveloper allows us to simply specify the directory "src\org\jdeveloper" and from looking at the package statements of the java source files located in "src\org\jdeveloper", it derives that the source path is "src". In other tools, you

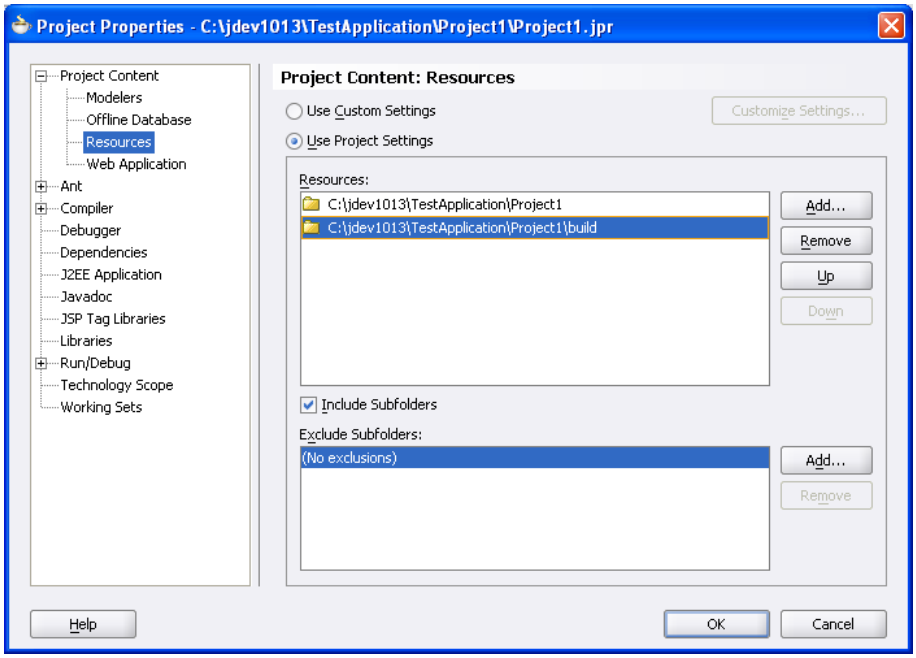
would be required to add a source path entry of "src", include package org.jdeveloper and also exclude the packages of mypackage and org.jdeveloper.test. By allowing JDeveloper to derive the source path for your code, you only need to specify the directory containing your source files without the bother of the underlying Java concept of a source path.

Resources

In addition to Java content, JDeveloper supports the concept of a path for resources of your project. A resource may be properties files, build scripts, or even specifications created in a word processor or other tool. The difference between Resources and Java Content is that no source path is derived from the files located in the Resources content set; therefore, these paths are not passed to the Java compiler. By default, JDeveloper adds the directory containing your project file as a non-recursive entry in the Resource content set.

If you are using Ant and create your build script in a directory other than the same directory as your project, you will need to add the directory containing the build.xml file to the Resources content set of your project.

Following is an example of a Resource content set that includes an additional directory for inclusion of an Ant build script. The Ant build.xml file is located in the build directory under the directory containing the JDeveloper project file.



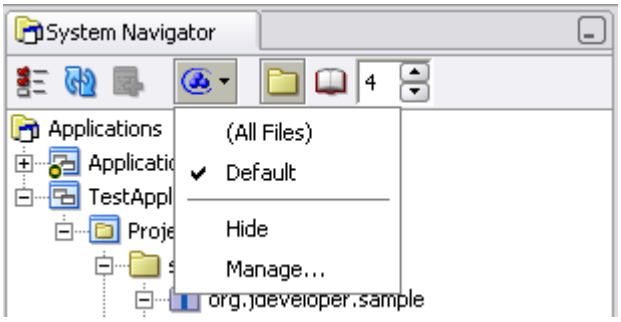
Working Sets

JDeveloper also supports the concept of a working set. A working set is a filter that is applied to the list files that you see in the navigator in JDeveloper, but has no effect on the actual list of files contained within your project. The working set is a simple filter on your view that does not impact the project. If you do a refactoring operation, JDeveloper will search the entire project-not simply the working set-as JDeveloper does not limit refactoring operations to a working set.

Additionally, a working set is not stored in the project file, and is therefore a setting that is unique to your copy of JDeveloper and not shared with the team.

You can access the working set settings from the Project Properties dialog (located under the Tools Menu or context menu in the Navigator) and from the working set icon located in the System Navigator.

By default, you are already using a working set that is created automatically for you. It is aptly named "Default", and enabled. To disable the default working set, you need to select "(All Files)" from the working sets dropdown located on the System Navigator Toolbar. At this point it will be checked instead of "Default".



Creating Working Sets

From the "Manage" option of the working sets dropdown or the Project Properties dialog, you can create a new working set. The "New " button on the working sets page will create a new working set. After you name the working set, you need to customize the list of files contained in it. From the "Files" tab you deselect individual files to remove them from the list of files contained in the working set.

The "Patterns" tab allows you to specify a series of Include and Exclude patterns that use the same wildcard syntax as Ant to create the list of files contained within the working set.

If you use the Patten based approach, it is important to note that processing for a file will stop at the first pattern match and the filters are processed from top to bottom.

Managing Working Sets

After you have created your working set, you can add or remove files at any time. You use the same steps to manage a working set that you did when you created it initially.

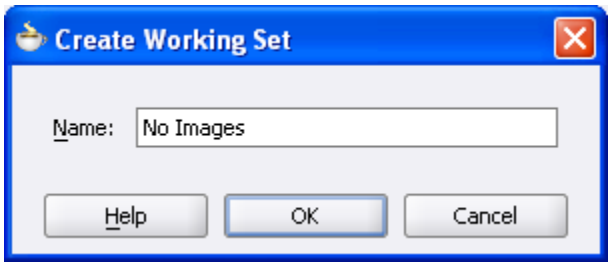
You will notice that as you use the checkboxes in the "Files" tab of the working sets page, entries in the "Patterns" tab are automatically added for you. At some point, your Patterns may become overly complex as you add and remove the same files over and over again from the check boxes. If this happens, you should remove some of the entries from the "Patterns" tab to simplify the list. Additionally, you can remove all of the entries and recreate your working set.

Guidelines for using Working Sets

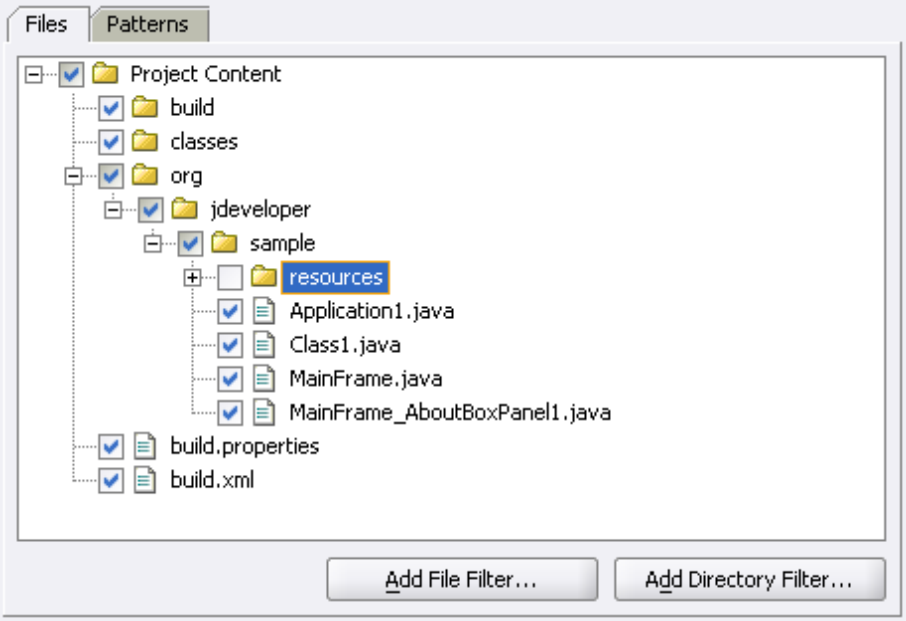
As mentioned before, working sets are not contained within the project file itself and, therefore, are not shared with the team. Working sets should not be used to remove files from the project, but rather keep them out of your view. Most operations dealing with projects will still continue to operate on files that are hidden from view by your working set.

Examples

Here we create a working set name "No Images" which will filter out all of our Image Files.

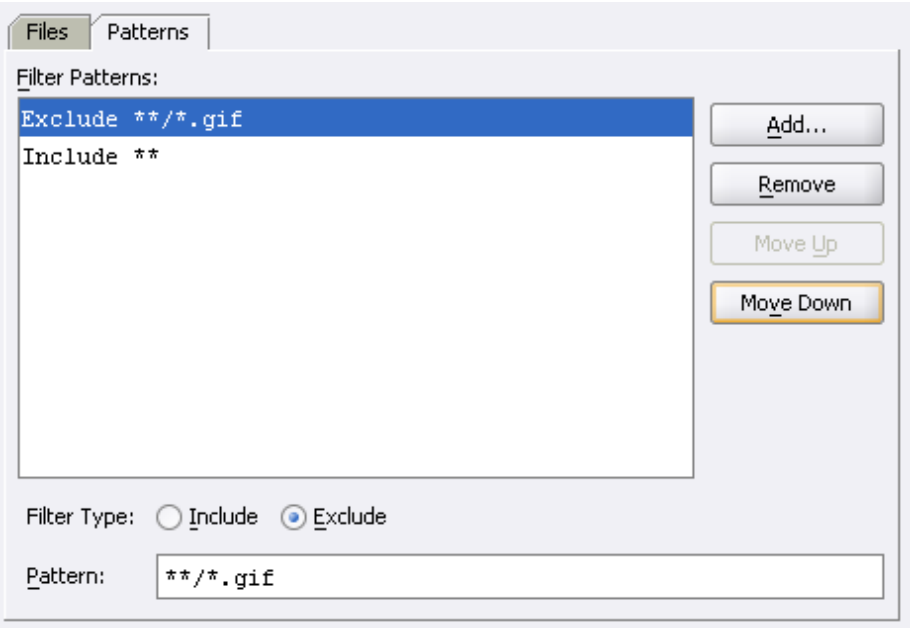


There are two different approaches that you may take to accomplish this task. The first is to remove the resources directory, where all of the GIF files are currently located, from the working set. You can do this from the "Files" tab of the working sets page in the Project Properties dialog. You can also exclude all GIF files by clicking on the "Add File Filter…" button and entering .gif as the file type to exclude.



Additionally, you could add a filter in the "Patterns" tab. To filter out all of the files ending in gif, you must add a filter of **/*.gif. The ** in the pattern tells JDeveloper to recurse through directories and match on an ending pattern of *.gif, meaning all files ending in .gif.

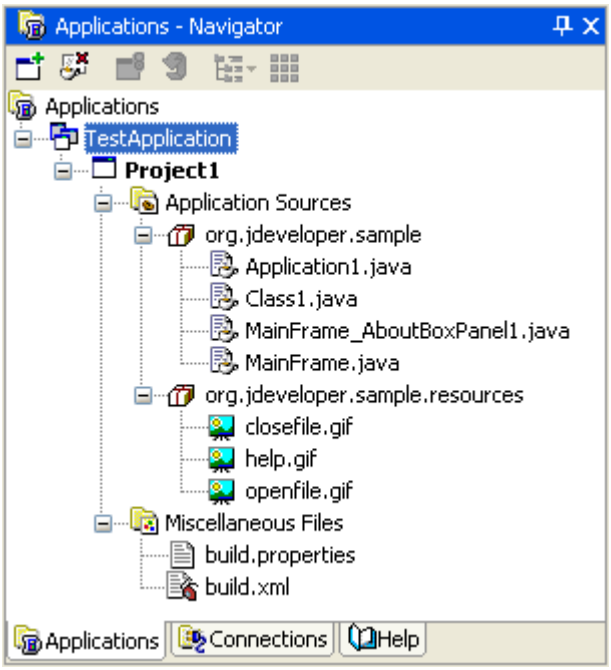
More information on the patterns used in JDeveloper and Apache Ant can by found in the section for dirtasks located in the Apache Ant Manual.
http://ant.apache.org/manual/dirtasks.html



You can get started with CVS from either http://www.cvshome.org or http://www.cvsnt.org. Both sites have versions of CVS for Windows, Linux, Mac OS X, and various other platforms.

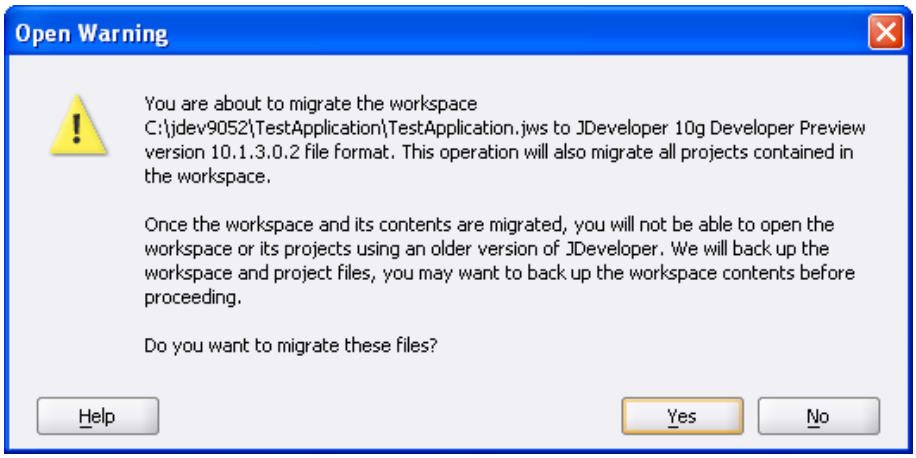
MIGRATING PROJECTS TO JDEVELOPER 10.1.3

JDeveloper will automatically migrate your project and application files created with an older version of the tool. Starting with JDeveloper 10.1.3, a backup file with an extension of ".bak" is generated for each of the project and application files, and this information is written to the Messages log page of the IDE. Although JDeveloper will create backup files, Oracle recommends that you make a complete backup of your sources, projects, and application files as a precautionary measure. This backup could either be a tag or label assigned to the files in your source control system, or a zip or tar archive created from your source files if you are not using a source control system. Additionally, if you are not currently using any source control system, you may wish to investigate CVS. CVS is a free and open source product that is an excellent introduction to the concepts of source code control and management. In our example, we will use our original example where all of the files in the org.jdeveloper.test and mypackage packages are excluded by default. Following is a screenshot of our original application.



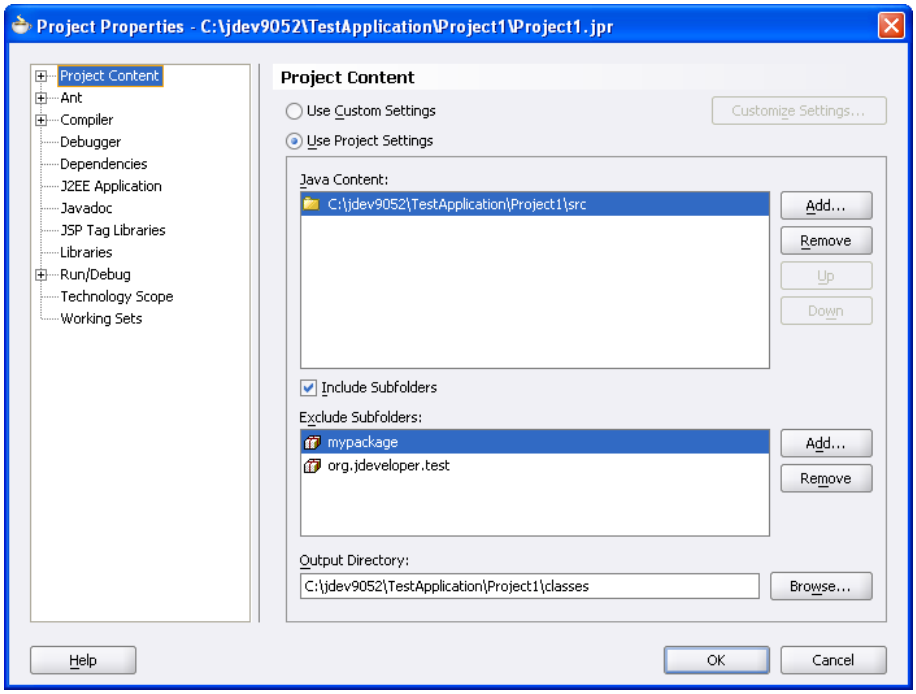
Automatic Migration

When you open a Project or Application (Workspace) from an older version of JDeveloper in your JDeveloper 10.1.3 environment, you are prompted with a warning dialog to ensure that you actually intend to migrate the 10.1.2 project and application to the new JDeveloper 10.1.3 format. After you confirm the migration, the Project files and Application files are automatically migrated to the current version and opened by the IDE.

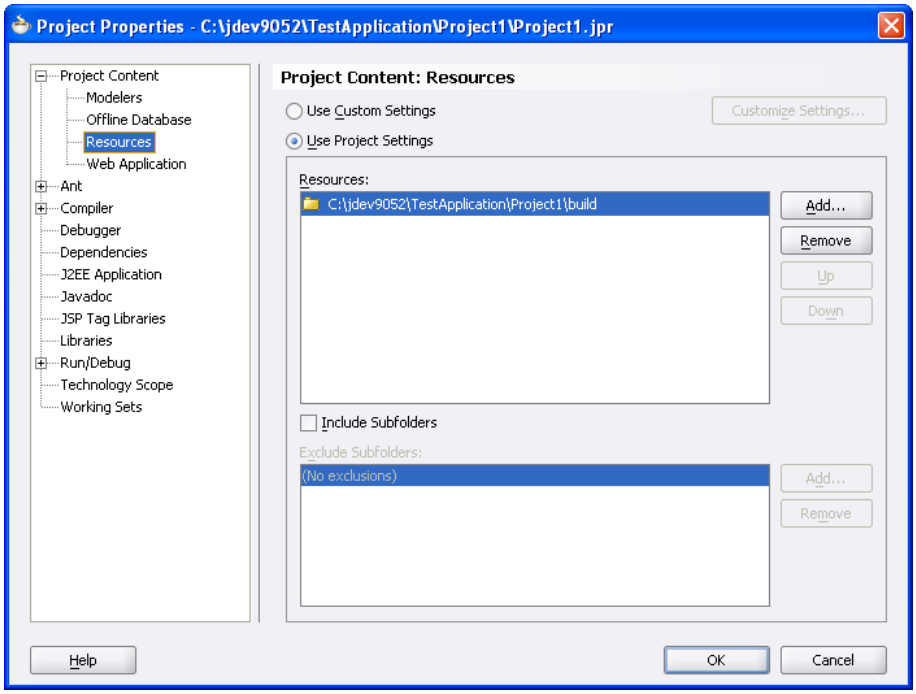


You should open the Project Properties and inspect the settings for your newly migrated project to confirm that all of the settings are as expected. In our example, JDeveloper's migration correctly derived that we did not want any of the files located in the following subdirectories under our source root.

- mypackage
- org\jdeveloper\test



Also since our 10.1.2 project contained a couple of files, which are not Java source files, JDeveloper also derived the need for a resource content set containing our "build" directory from our 10.1.2 project. The resources content set is for files that are not Java source files and, therefore, should never be passed on a derived source path entry to the Java compiler. Ant build files are a prime example of the type of files that should reside in one of the directories contained in the resources content set.



Manual Adjustments

In our example, each directory either contained no files that comprised our JDeveloper project, or every file in the directory was a member of our project. In some cases, only some of the files in a given directory may actually be included in your project. If this is the case in your environment, you will need to make some manual adjustments to your project after migration to ensure that unwanted files will not be in your project.

During migration, if none of the files for a directory were included in the 9.0.5 project, JDeveloper will exclude that directory. If at least one file in the directory is a member of the 9.0.5 project, then no exclusion will be added to the resulting 10.1.3 project.

There are several approaches to handling unwanted files in 10.1.3 post migration. If the files are not in a state that may be compiled, you should rename them to change the extension from being java, or move them to another directory that is not part of the Java content set. Alternatively, you could place these files in a directory located in the resources content set. If the files are compilable, but should just remain hidden, you could create a working set that hides these files from view. Whatever approach you take to handle these files depends on your requirements and your unique environment. The important thing to realize is that the file system is the source of truth for determining the list of files that comprise your project in JDeveloper 10.1.3.

BEST PRACTICES FOR MIGRATION

Now that you are aware of the differences in projects between older versions of JDeveloper and the JDeveloper 10.1.3 release, it is often helpful to mitigate some of the issues before migration. Here are a couple of pre-migration tasks to ensure your migration is as easy as possible.

- Back up your entire source tree before migration.
- Ensure that every file in a given directory is in your project, or that the directory contains no files that are in your project. This will help to make sure you do not gain any "extra" files post-migration.

FUTURE IMPROVEMENTS

In the production release of JDeveloper 10.13, the project filtering will be improved to allow file level filters, as well as the current directory only exclusion found in the 10.1.3 preview release. Further improvements for automatically creating a working set from the results of an Audit or Find Usages command are also planned.

CONCLUSION

Moving from a manual list of files that comprise projects to a file system centric view is a fundamental difference between JDeveloper 10.1.3 and earlier releases. It is important to how the contents of project files is calculated, as well as how source path entries are derived in JDeveloper to understand how migration issues can arise. These changes drastically reduce the complexity of the project files, and enable JDeveloper projects to scale larger teams of developers. Taking some pre-migration steps and understanding the fundamental changes to projects will ensure a smooth transition to the latest version of JDeveloper.

White Paper Title
February 2005
Author: Robert Clevenger
Contributing Authors:

Oracle Corporation
World Headquarters
500 Oracle Parkway
Redwood Shores, CA 94065
U.S.A.

Worldwide Inquiries:
Phone: +1.650.506.7000
Fax: +1.650.506.7200
oracle.com

Copyright © 2005, Oracle. All rights reserved.
This document is provided for information purposes only
and the contents hereof are subject to change without notice.
This document is not warranted to be error-free, nor subject to
any other warranties or conditions, whether expressed orally
or implied in law, including implied warranties and conditions of
merchantability or fitness for a particular purpose. We specifically
disclaim any liability with respect to this document and no
contractual obligations are formed either directly or indirectly
by this document. This document may not be reproduced or
transmitted in any form or by any means, electronic or mechanical,
for any purpose, without our prior written permission.
Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective owners.