Overview

> This is deprecated information!

(Note that this content is from 2007 and may no longer be correct or necessary. See links below and use the site search for finding info on deployment issues.)

(Note that the following links to the most up to date Mac Deployment information - Deploying on Mac OS X 10.6 (Snow Leopard))

Xcode expects to take a given set of files, as listed in your Groups & Files pane and from them build a finished product or set of products such as an application, a framework, a library or a command line tool.

It's worth noting here that the items represented in this pane are of four types: files, directories, packages and groups. Files are just that, files of various types with extensions indicating the type, such as .java, .jpg, .gif, .html. Directories represent real directories somewhere in your file system and are represented with a blue folder icon. Package is a Mac OX X term for a directory of files and subdirectories that are to be recognized as a unit both by the Finder and by Xcode. They are often represented with an icon that is indistinguishable from a file icon. If Xcode isn't familiar with the kind of package represented, it displays them as blue folders. Groups are the strange items in this categorization and are pictured as yellow folders. THEY DON'T EXIST AT ALL in the file system. They are purely organizing entities for your use in Xcode. You can group files together in an Xcode Group in the Groups and Files pane that are actually distributed all over the file system of your computer. You use them in the Groups & Files pane simply to group together those items of any of the four types just described that you feel are related somehow.

Getting back to Xcode's products, for a typical, simple, non-WebObjects project, the files would comprise Java or Objective C source files which would be compiled to intermediate object files and linked into a load module (for Objective C) or a .jar file (for Java) for execution. That file, created for execution is what I'm calling the product. An Xcode target is the set of files along with the instructions necessary to build such a product. It will be brought into existence by Xcode build processes.

The situation is complicated slightly in the case of WebObjects in that good practice suggests something called a "split install". That is, for WebObjects projects, Xcode expects to build at least two products. One of these will be the WebObjects application which will be served by the application server. The other will be a package of static HTML pages, graphics files (.jpg and .gif files, among others) and other files that can be served directly by Apache as static resources.

The files that you add to the project must be identified as belonging to the building of one or the other of these two products. The third target, which carries the same name as your project, is something called an aggregate target. It's sole purpose is to group the other two targets into a single build operation by setting the dependencies such that the aggregate target depends on the other two. That way, by building the aggregate target, the other two targets will automatically be built together. However, you can also ask Xcode to build either of these other two targets individually.

Although you should have added the static files to your project so that they are included in the project directory, Xcode will build two packages, one for the application server and one for the web server and will either create (e.g. by compiling) or copy the appropriate files to the appropriate package. If your project were named "MyProject" and were built for deployment on Mac OS X, these packages would have the following structures:

For the application server:

```
MyProject.woa
   MyProject - a Unix shell script to which control is initially passed and
which sets up the execution environment under Unix
   MyProject.CMD - a DOS command script to which control is initially
passed and which sets up the execution environment under DOS/Windows
   WOBootstrap.jar - a jar file containing a single class called
JavaArchiveFilter that implements the java.io.FilenameFilter interface
   Contents - a directory containing all of the files necessary to support
the execution of the WO app
        Info.plist - a property list file for the application
       MacOS/
           MyProject - a Unix shell script identical to that of the same
name at the top level of the project directory
           MacOSClassPath.txt - text file containing the derived Java
ClassPath for the project
       PkgInfo - a text file containing the four letter package type (APPL
= application) and the application's four letter file types
       Resources/ - a directory containing the WOComponents and other
resources in the Resources group of the Groups & Files pane
       UNIX/ - a directory similar to the MacOS directory but used for
deployment on Unix systems
       Windows/ - a directory similar to the MacOS directory but used for
deployment on Windows systems
       pbdevelopment.plist - an XML file with a single element whose key is
PBXProjectSourcePath and whose value is implied by the name.
```

For the Web Server:

```
MyProject.woa
   Contents - a directory containing all of the files necessary to support
the execution of the WO app
       WebServerResources/
           Hierarchy of files and folders containing static web resources.
```

Although the packages are very different, they are identically named, have similar structures and are merged in development. They are typically ONLY separated like this for a split install. You typically put the application server package in /Library/WebObjects/Applications (or a local equivalent) and you typically put the WebServer package in /Library/WebServer/Documents/WebObjects.

You can create these packages independently in various ways. I've found it easier, recently (because of various examples of Xcode intransigence) to simply build the project as I would for development and physically separate the two for deployment. Other people have various incantations to make a split install work reliably and I'm sure they will chime in here.

Other Links:

- Standard Deployment
- Tomcat Deployment
- Split Install Deployment
- Remote deployment with Ant and ssh
- Logging