About the Properties file

Introduction

Most applications need a place to hold application configuration information. In java, a file with simple entries in traditional java.util.Properties format can be read and merged with the System properties, usually at application launch time. The traditional (as distinct from the XML format supported by java 1.5) simple format of a java properties file is like this:

```
myproperty=12345
myotherproperty=This is a sentence
```

WebObjects manages application preferences using a similar mechanism implemented by its com.webobjects.foundation.NSProperties class. The file format is as per the spec. In many references you will see specs say that a certain file is in java.io.Properties file format, however if I am not mistaken, they really mean the java.util.Properties, since even as far back as Java 1.1 API, there was no such class as java.io.Properties.

In any case, WebObjects directly supports the use of Java 1.4 style properties entries in the file named Resources/Properties in the woa and framework bundle format.

Using Properties in WebObjects

For a traditional WebObjects application, Properties are read first from the framework bundles and finally from the application bundle. Hence frameworks can have default values in their Properties file and those framework configuration properties can be **over-ridden** in the application's Properties file

No coding is needed to use property entries in the Properties file. WebObjects automatically reads them in at application launch time and merges them with the system properties, so they can be accessed using System.getProperty

Enhanced Properties management in WOnder

Project WOnder enhances the use of the traditional WebObjects Properties file in a number of ways:

ERXProperties

WOnder introduces the class ERXProperties that provides enhancements to the kind and format of Properties files

Configurability

Firstly, WOnder makes useful settings available as Properties which are not available out of the box as simple Properties entries in traditional WebObjects applications.

For example:

- EOModel connection dictionary parameters for each model
- log4j configuration properties
- Prototype model loading order

User Properties

If your user name is 'joe', then in your WebObjects application, you can create a file named Properties.joe which is read in last when joe launches his WebObjects app in his Eclipse development enviromnent. This is useful for development teams. Each team member can have properties specific to his /her own development enviromnent such as smtp server, logging properties, database connection dictionary settings, etc.

Derived Properties

WOnder supports the use of @@ delimiters so that certain properties can depend on previously defined properties.

Derived Properties example

```
## Set the smtp server host one time here... and the rest get it thru
#+ use of derived propertiesw calculated on the fly when loaded into System properties.
app.smtpserver=mail.domain.com

## WebObjects smtp host setting
WOSMTPHost=@@app.smtpserver@@

## Setting for a log4j smtp appender named 'myMail'
log4j.appender.myMail.SMTPHost=@@app.smtpserver@@

## Setting for Wonder's ERJavaMail framework
er.javamail.smtpHost=@@app.smtpserver@@

## Setting for Sun's javamail library
mail.smtp.host=@@app.smtpserver@@
```

Accessing properties in WOnder

Use ERXProperties static methods to get system properties from your Properties files and it offers a bunch of convenience methods to return values coerced into thyeir intended object types. It even handles arrays .... look at the source or API for ERXProperties for more details.

Conclusion

So, simply putting application and logging properties into the Properties file means that all your configuration is in one place **and** each member of the development team can override those deployment settings with their own user based Properties files. Useful when they want to add DEBUG logging on stuff that other team members are not interested in.