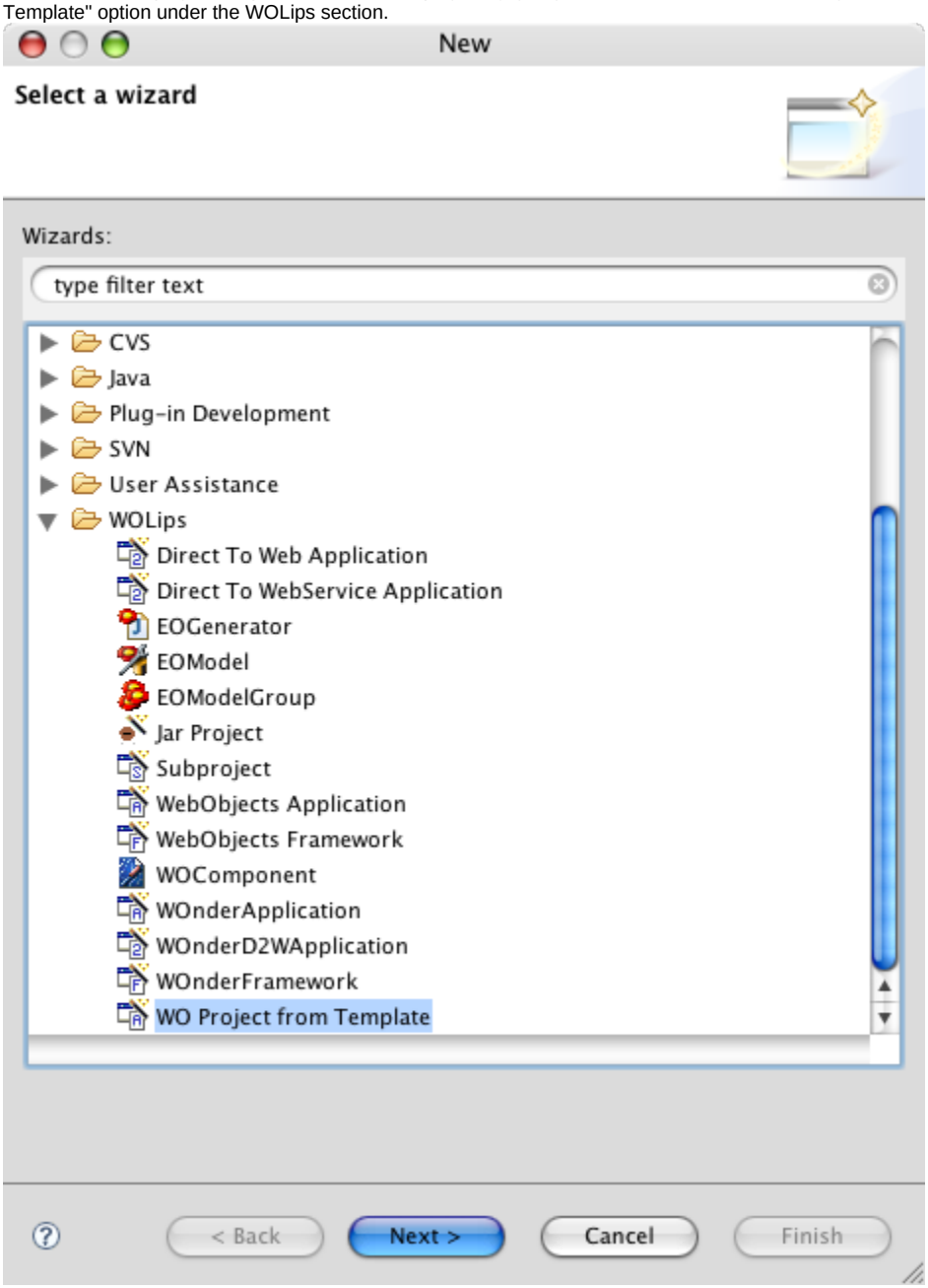
Custom Project Templates

Introduction

WOLips 3.3+ supports the definition of custom project templates. These templates support creating new types of WebObjects projects beyond the ones that ship with WOLips.

To use a custom project template, create a new project by going to the "File" menu and selecting "New -> Other..." then select the "WO Project from Template" option under the WOLips section.



Creating a Template

Templates are basically like a regular project folder, but some of the contents are modified to use the variables defined in the template.xml file to change their names or values in their contents. The modified files within the template are processed using the Velocity Template Engine.

The declaration of templates is very simple. The template system looks for folders in the following locations (in the following order):

1. The ProjectTemplates folder inside of the WOLips templateengine plugin jar
2. /Library/Application Support/WOLips/Project Templates
3. YourHomeDir\Documents and Settings\Application Data\WOLips\Project Templates
4. YourHomeDir\Documents and Settings\AppData\Local\WOLips\Project Templates
5. ~/Library/Application Support/WOLips/Project Templates

Project templates found later in the list of locations above will override those of the same name from earlier in the list. For instance, a "Wonder Application" template found in ~/Library will override a "Wonder Application" template from /Library.

The name that the template shows up as in the New Project wizard is controlled by one of two things:

1. The template's folder name
2. The name attribute of the template tag in the template.xml file inside the template folder (if present, this overrides the value set by the template folder name).

For instance, if you create a folder named "~/Library/Application Support/WOLips/Project Templates/Wonder Application," the template system will offer a template named "Wonder Application" in the template selection dialog. If you then set the name attribute to "My Wonder Application", that is what you will see in the dialog.

If you want to look at the current templates that ship with WOLips, the best way to analyse them is by importing the source of WOLips from Subversion. The templates are located in *woproject/wolips/core/plugins/org.objectstyle.wolips.templateengine/ProjectTemplates*.

Here is an additional custom template for creating Wonder ERD2W Applications that you can view as an example.

Remember, if you copy an existing template to test it out you must rename both the folder **and** the name attribute of the template tag in the template.xml file otherwise whichever loads last will replace the first!

After creating a template folder, you can create a hierarchy of files and folders within that folder. When you create a project using this template, a copy of all of the files and folders in your template will be used to create the new project. It is up to you to declare *all* of the files within a project, including Eclipse project metadata files like .classpath. You can refer to the built-in project templates as a starting point for creating your own custom templates.

That's it! For creating static boilerplate templates, you're done.

Template Metadata and Template Inputs

For a static template, the simple process above is enough. However, it's a common requirement to have configuration options for project templates. The WOLips project template engine provides an easy way to declare these options.

After creating a template using the directions in the above section, you can additionally create a file named "template.xml" inside your project template folder. For instance, in the example above, you would create the file "~/Library/Application Support/WOLips/Project Templates/My Application Template /template.xml".

An example template.xml is below:

```
<?xml version="1.0" encoding="UTF-8"?>
  <template name = "My Application">
    <inputs>
      <input name = "basePackage" type = "Package">
        <question>Base Package?</question>
        <default>your.app</default>
      </input>
      <input name = "componentsPackage" type = "Package">
        <question>Components Package?</question>
        <default>your.app.components</default>
      </input>
      <input name = "servletDeployment" type = "Boolean">
        <question>Deploy to Servlet Container?</question>
        <default>false</default>
      </input>
      <input name = "webXML" type = "Boolean">
        <question>Autogenerate web.xml file?</question>
        <default>false</default>
      </input>
      <input name = "YourFavoriteColor" type = "String">
        <question>Your Favorite Color?</question>
        <options>
          <option name = "Red" value = "#FF0000"/>
          <option name = "Green" value = "#00FF00"/>
          <option name = "Blue" value = "#0000FF"/>
        </options>
        <default>#FF0000</default>
      </input>
    </inputs>
  </template>
```

If you don't set a "name" attribute in the "template" tag, the name of the Template Folder will be used. In this example the Template will show up as "My Application" in the list of available templates. If "name" handn't been defined, the it would show up as the folder name, which was "My Application Template".

Within a template, you can declare a single "inputs" node that can contain multiple "input" nodes. Each input node corresponds to a variable that will be presented to the user on the second page of the wizard. Each input specifies a "name" attribute, which will become the variable name of the input for later reference in the Velocity templates; and a "type" attribute which will determine the type of control presented to the user. They are:

- Boolean -> Check Box
- String -> Text Field
- Package -> Text Field
- Integer -> Spinner

Each input also contains a "question" node, whose value corresponds to the label of the control when displayed to the user. In the above example, the "servletDeployment" will display a checkbox to the user with the label "Deploy to Servlet Container?". Additionally, you can provide a "default" node that defines the default value of the variable. If a default is not specified, the default value will be null for all input types.

The package type is slight extension to the String type. For a variable declared as type Package, in addition to having your variable bound, you will also have a variable named "yourvariablename_folder" with replaces dots for slashes. For instance, if your variable is named "basePackage" you will also get a variable named "$basePackage_folder". If the user left the default of "your.app" in the basePackage field, then the two variables would be:

- basePackage = "your.app"
- basePackage_folder = "your/app"

Finally, the input system supports the declaration of enumerated types. By declaring an "options" node that contains an ordered set of "option" nodes, you can define the possible values that the user can provide. In the above example, the "YourFavoriteColor" input defines three options: Red, Green, and Blue. Each option node has a "name" attribute, which will be the value displayed to the user, and a "value" attribute, which will be the actual backing value of the selection. The value of the option should be of the type specified in the "type" attribute of the input. For instance, if you declare the input type to be "Integer," your option values should be integer values (in quotes).

Using Template Inputs

So now that you have template inputs defined, you need to be able to use them. The name used in the "name" attribute of your input node will be the name of the variable you can use in your template's files. For instance, in the example above, the Velocity variable "servletDeployment" will be bound to the boolean value corresponding to the user's selection, and can be used just like any other velocity variable. The Apache project provides a Velocity reference guide.

As an example, the .classpath file can be modified to automatically add a link to the JavaWOJSPServlet.framework if the user selects "Deploy to Servlet Container" in the the wizard.

```
<?xml version="1.0" encoding="UTF-8"?>
<classpath>
        <classpathentry kind="src" path="Sources"/>
        <classpathentry kind="con" path="WOFramework/JavaEOAccess"/>
        <classpathentry kind="con" path="WOFramework/JavaEOControl"/>
        <classpathentry kind="con" path="WOFramework/JavaFoundation"/>
        <classpathentry kind="con" path="WOFramework/JavaJDBCAdaptor"/>
        <classpathentry kind="con" path="WOFramework/JavaWebObjects"/>
        <classpathentry kind="con" path="WOFramework/JavaXML"/>
        <classpathentry kind="con" path="org.eclipse.jdt.launching.JRE_CONTAINER"/>
#if ($servletDeployment)
        <classpathentry kind="con" path="WOFramework/JavaWOJSPServlet"/>
#end
        <classpathentry kind="output" path="bin"/>
</classpath>
```

In addition to variables inside of Velocity templates, you can also use template variables in folder and file names. However, because $ is not allowed on some filesystems, we instead surround the variable names with "__" (for instance $someVariable would be "__someVariable__" in the filename or path). As an example, the template above has an input named "basePackage" (of type Package) which, as we learned above, creates two variables: "basePackage" and "basePackage_folder". Since we are going to be using this variable to name (and define the path) of a folder in the Sources directory, we need to use the alternate version of the variable. Create a new folder in the Sources directory of the project templaet and instead of using "${basePackage_folder}" as the file name, we must use "__basePackage_folder__".

Special Circumstances

Flagging files to be skipped by Velocity

If there are files in your template that should not be processed by Velocity as the project is being setup, you will need to end the file name with "__binary" which will cause Velocity to strip the "__binary" off the file name, but skip processing the contents of the file. Examples of files that you'd want to flag are:

- EOGenerator Templates
- Custom builder .launch files.

Using keypaths

If you need to call a method on a given variable, you need to include the "()" at the end of the method name. An eample is in the build.properties file. In order to get the lowercase version of the project name, you have to call "${projectName.toLowerCase()}"

Happy templating!