Expression Examples

The [previous post](http://codeandme.blogspot.com/2012/04/source-provider-property-tester-and.html) about the Expression Framework might be a bit confusing. Therefore I will provide some expression examples for eclipse built in source providers and tester and for the custom property tester provided in the previous post.  
  
Source code for this tutorial is available on googlecode as a [single zip archive](http://codeandme.googlecode.com/svn/trunk/blog/Code%20&%20Me%20Blog/core_expressions.zip), as a [Team Project Set](http://codeandme.googlecode.com/svn/trunk/blog/Code%20&%20Me%20Blog/core_expressions.psf) or you can [checkout the SVN](http://codeandme.googlecode.com/svn/trunk/blog/core_expressions/) projects directly.  
  
**Preparations**  
  
Create a new *Plug-in Project* called *com.example.coreexpressions.samples*. Add dependencies to *org.eclipse.ui*,*org.eclipse.core.runtime* and *com.example.coreexpressions*. Set the content of *plugin.xml* to

[view plainprint?](http://codeandme.blogspot.in/2012/04/expression-examples.html)

1. **<?xml** version="1.0" encoding="UTF-8"**?>**
2. **<?eclipse** version="3.4"**?>**
3. **<plugin>**
4. **<extension**
5. point="org.eclipse.ui.menus"**>**
6. **<menuContribution**
7. allPopups="false"
8. locationURI="toolbar:org.eclipse.ui.main.toolbar"**>**
9. **<toolbar**
10. id="com.example.coreexpressions.toolbar1"**>**
11. **</toolbar>**
12. **</menuContribution>**
13. **</extension>**
14. **<extension**
15. point="org.eclipse.ui.commands"**>**
16. **<command**
17. defaultHandler="com.example.coreexpressions.samples.commands.Login"
18. id="com.example.coreexpressions.commands.login"
19. name="login"**>**
20. **</command>**
21. **<command**
22. defaultHandler="com.example.coreexpressions.samples.commands.Logout"
23. id="com.example.coreexpressions.commands.logout"
24. name="logout"**>**
25. **</command>**
26. **</extension>**
28. **</plugin>**

Add a new class *com.example.coreexpressions.samples.commands.Login*

[view plainprint?](http://codeandme.blogspot.in/2012/04/expression-examples.html)

1. **package** com.example.coreexpressions.samples.commands;
3. **import** org.eclipse.core.commands.AbstractHandler;
4. **import** org.eclipse.core.commands.ExecutionEvent;
5. **import** org.eclipse.core.commands.ExecutionException;
6. **import** org.eclipse.core.commands.IHandler;
7. **import** org.eclipse.ui.ISourceProvider;
8. **import** org.eclipse.ui.PlatformUI;
9. **import** org.eclipse.ui.services.ISourceProviderService;
11. **import** com.example.coreexpressions.ExampleSourceProvider;
13. **public** **class** Login **extends** AbstractHandler **implements** IHandler {
15. @Override
16. **public** Object execute(ExecutionEvent event) **throws** ExecutionException {
18. ISourceProviderService service = (ISourceProviderService) PlatformUI.getWorkbench().getService(
19. ISourceProviderService.**class**);
20. ISourceProvider provider = service.getSourceProvider(ExampleSourceProvider.CURRENT\_USER);
21. **if** (provider instanceof ExampleSourceProvider)
22. ((ExampleSourceProvider) provider).setCurrentUser("John Doe");
24. **return** **null**;
25. }
26. }

and another class *com.example.coreexpressions.samples.commands.Logout*

[view plainprint?](http://codeandme.blogspot.in/2012/04/expression-examples.html)

1. package com.example.coreexpressions.samples.commands;
3. import org.eclipse.core.commands.AbstractHandler;
4. import org.eclipse.core.commands.ExecutionEvent;
5. import org.eclipse.core.commands.ExecutionException;
6. import org.eclipse.core.commands.IHandler;
7. import org.eclipse.ui.ISourceProvider;
8. import org.eclipse.ui.PlatformUI;
9. import org.eclipse.ui.services.ISourceProviderService;
11. import com.example.coreexpressions.ExampleSourceProvider;
13. public class Logout extends AbstractHandler implements IHandler {
15. @Override
16. public Object execute(ExecutionEvent event) throws ExecutionException {
17. ISourceProviderService service = (ISourceProviderService) PlatformUI.getWorkbench().getService(
18. ISourceProviderService.class);
19. ISourceProvider provider = service.getSourceProvider(ExampleSourceProvider.CURRENT\_USER);
20. if (provider instanceof ExampleSourceProvider)
21. ((ExampleSourceProvider) provider).setCurrentUser(null);
23. return null;
24. }
25. }

For our tests we will create commands in the defined toolbar and add a *visibleWhen* expression. As default *commandId* I will use the predefined org.eclipse.ui.file.exit.  
  
**Test 1: Enable for a specific view**  
  
Our first expression will reveal a toolbar item when a specific view is activated.

[view plainprint?](http://codeandme.blogspot.in/2012/04/expression-examples.html)

1. **<command**
2. commandId="org.eclipse.ui.file.exit"
3. label="Test1"
4. style="push"**>**
5. **<visibleWhen**
6. checkEnabled="false"**>**
7. **<with**
8. variable="activePartId"**>**
9. **<**equals
10. value="org.eclipse.ui.navigator.ProjectExplorer"**>**
11. **</equals>**
12. **</with>**
13. **</visibleWhen>**
14. **</command>**

We need to set *checkEnabled* on the *visibleWhen* element to *false*, otherwise the expression will not be active. The [with](http://wiki.eclipse.org/Platform_Expression_Framework" \l "with" \t "_blank)section uses activePartId as source. It is a string containing the ID of the active view/editor (see [description](http://wiki.eclipse.org/Command_Core_Expressions#Variables_and_the_Command_Framework)).  
  
The [equals](http://wiki.eclipse.org/Platform_Expression_Framework#equals) section compares the source value with *org.eclipse.ui.navigator.ProjectExplorer*, returning true when the Project Explorer View is active.  
  
Run your Plug-in as a new Eclipse application and activate the Project Explorer to see the toolbar entry appear/disappear.  
  
**Test 2: Enable when Projects are selected**  
  
  
Next we want a button to be visible when at least one project is selected.

[view plainprint?](http://codeandme.blogspot.in/2012/04/expression-examples.html)

1. **<command**
2. commandId="org.eclipse.ui.file.exit"
3. label="Test2"
4. style="push"**>**
5. **<visibleWhen**
6. checkEnabled="false"**>**
7. **<with**
8. variable="selection"**>**
9. **<**iterate
10. operator="or"**>**
11. **<**adapt
12. type="org.eclipse.core.resources.IProject"**>**
13. **</adapt>**
14. **</iterate>**
15. **</with>**
16. **</visibleWhen>**
17. **</command>**

Now we use the *selection* source provider, which provides the current selection. The [iterate](http://wiki.eclipse.org/Platform_Expression_Framework#iterate) operator is nice as it tests all elements of a collection. By setting *operator* to *or* we define that at least one element of the collection has to pass the test.  
  
  
There exists an [instanceof](http://wiki.eclipse.org/Platform_Expression_Framework" \l "instanceof" \t "_blank) operator which we could use to check against org.eclipse.core.resources.IProject. Unfortunately not all projects implement that interface. Instead they use the adapter framework. Therefore we use the [adapt](http://wiki.eclipse.org/Platform_Expression_Framework#adapt) operator to see if the element can be adapted to an IProject. Of course this works for all classes implementing the interface directly too.  
  
  
**Test 3: Enable when exactly 2 txt Files are selected**  
  
  
Now for something more complex. A button shall be visible when exactly two resources are selected and both have an extension of *.txt*.

[view plainprint?](http://codeandme.blogspot.in/2012/04/expression-examples.html)

1. **<command**
2. commandId="org.eclipse.ui.file.exit"
3. label="Test3"
4. style="push"**>**
5. **<visibleWhen**
6. checkEnabled="false"**>**
7. **<with**
8. variable="selection"**>**
9. **<and>**
10. **<iterate**
11. operator="and"**>**
12. **<test**
13. property="org.eclipse.core.resources.extension "
14. value="txt"**>**
15. **</test>**
16. **</iterate>**
17. **<**count
18. value="2"**>**
19. **</count>**
20. **</and>**
21. **</with>**
22. **</visibleWhen>**
23. **</command>**

The property tester for file extensions is [described in the documentation](http://wiki.eclipse.org/Command_Core_Expressions#Variables_and_the_Command_Framework). The [count](http://wiki.eclipse.org/Platform_Expression_Framework#count) operator counts the elements within a collection.  
  
  
**Test 4: Login/Logout**  
  
  
Let's add two more buttons for a very basic user login.

[view plainprint?](http://codeandme.blogspot.in/2012/04/expression-examples.html)

1. **<command**
2. commandId="com.example.coreexpressions.commands.login"
3. label="Login"
4. style="push"**>**
5. **<visibleWhen**
6. checkEnabled="false"**>**
7. **<with**
8. variable="com.example.coreexpressions.currentStatus"**>**
9. **<or>**
10. **<equals**
11. value="startup"**>**
12. **</equals>**
13. **<equals**
14. value="logged off"**>**
15. **</equals>**
16. **</or>**
17. **</with>**
18. **</visibleWhen>**
19. **</command>**
20. **<command**
21. commandId="com.example.coreexpressions.commands.logout"
22. label="Logoff"
23. style="push"**>**
24. **<visibleWhen**
25. checkEnabled="false"**>**
26. **<with**
27. variable="com.example.coreexpressions.currentUser"**>**
28. **<equals**
29. value="John Doe"**>**
30. **</equals>**
31. **</with>**
32. **</visibleWhen>**
33. **</command>**

We use dedicated commands that will set/reset the current user of our source provider from the [previous post](http://codeandme.blogspot.com/2012/04/source-provider-property-tester-and.html). The different source providers for both buttons do not necessarily make sense - we could achieve the same with just one source. Its just to give you an idea how it works.  
  
  
We use our custom property tester to verify the current logon state and the current user.

Source Provider, Property Tester and the Expressions Framework

The expressions framework provides powerful means to declare expressions via XML and query them at runtime. Such expressions can be used without loading the defining Plug-in. You can find them in command/menu enablements. There you find visibleWhen, enabledWhen, activeWhen expressions. But you also can use them for your own needs.  
  
In this example we will create our own Source Provider with a custom property tester (don't worry, I will explain this right ahead).  
  
There is good [documentation](http://wiki.eclipse.org/Platform_Expression_Framework) available for further reading.  
  
Source code for this tutorial is available on googlecode as a [single zip archive](http://codeandme.googlecode.com/svn/trunk/blog/Code%20&%20Me%20Blog/core_expressions.zip), as a [Team Project Set](http://codeandme.googlecode.com/svn/trunk/blog/Code%20&%20Me%20Blog/core_expressions.psf) or you can [checkout the SVN](http://codeandme.googlecode.com/svn/trunk/blog/core_expressions/) projects directly.    
  
**Introduction**  
  
The expression framework can create/evaluate boolean expressions - something that in the end evaluates to *true* or *false*. To evaluate we need an object to work with (a source) and operators which return a boolean value.   
  
Eclipse already defines commonly used [sources](http://wiki.eclipse.org/Command_Core_Expressions#Variables_and_the_Command_Framework) and [operators](http://wiki.eclipse.org/Platform_Expression_Framework#Operators_of_the_expressions_framework).  
  
A simple example of an expression looks like this:

[view plainprint?](http://codeandme.blogspot.in/2012/04/source-provider-property-tester-and.html)

1. **<visibleWhen>**
2. **<and>**
3. **<systemTest**
4. property="os.name"
5. value="Linux"**>**
6. **</systemTest>**
7. **<systemTest**
8. property="os.arch"
9. value="i386"**>**
10. **</systemTest>**
11. **</and>**
12. **<visibleWhen>**

*and* is a basic operator that - of course - performs a boolean and operation. *systemTest* queries a java system property and compares it to a given value. So this expression evaluates to true only if we are working on an i386 linux machine.  
  
Our source is not explicitely defined in the expression as the operator queries a static source *System.getProperties()*.  
  
**Step 1: Creating a SourceProvider**  
  
For most expressions we will need sources that contain modifiable content. Use the [predefined sources](http://wiki.eclipse.org/Command_Core_Expressions#Variables_and_the_Command_Framework) to get the active editor, the active selection or similar workbench related items. In this tutorial we will create a custom source to denote the current user.  
  
Create a new *Plug-in Project* called *com.example.coreexpressions*. In the *plugin.xml* create a new extension for*org.eclipse.ui.services* and add a new *sourceProvider* to it. Create two new *variables* for the sourceProvider: set *name* to*com.example.coreexpressions.currentStatus* and *com.example.coreexpressions.currentUser* and leave *priorityLevel* set to*workbench*.  
  
Now implement the sourceProvider by creating a class *com.example.coreexpressions.ExampleSourceProvider*:

[view plainprint?](http://codeandme.blogspot.in/2012/04/source-provider-property-tester-and.html)

1. **package** com.example.coreexpressions;
3. **import** java.util.HashMap;
4. **import** java.util.Map;
6. **import** org.eclipse.ui.AbstractSourceProvider;
7. **import** org.eclipse.ui.ISources;
9. **public** **class** ExampleSourceProvider **extends** AbstractSourceProvider {
11. **public** **static** **final** String CURRENT\_STATUS = "com.example.coreexpressions.currentStatus";
12. **public** **static** **final** String CURRENT\_USER = "com.example.coreexpressions.currentUser";
14. **private** Object mUser = **null**;
15. **private** String mStatus = "startup";
17. **public** ExampleSourceProvider() {
18. }
20. @Override
21. **public** **void** dispose() {
22. }
24. @Override
25. **public** Map getCurrentState() {
26. HashMap<String, Object> map = **new** HashMap<String, Object>();
27. map.put(CURRENT\_USER, mUser);
28. map.put(CURRENT\_STATUS, mStatus);
30. **return** map;
31. }
33. @Override
34. **public** String[] getProvidedSourceNames() {
35. **return** **new** String[] { CURRENT\_USER, CURRENT\_STATUS };
36. }
37. }

We need to implement two methods:

* *getProvidedSourceNames()*  
  returns an array of source identifiers (variables). These are the same we already defined as variables in the plugin.xml
* *getCurrentState()*  
  returns a Map<String, Object> wih entries for each defined variable.

**Step 2: Creating a property tester**  
  
Eclipse provides a generic expression operator *test* which allows to register custom tests.  
  
Create a new extension for *org.eclipse.core.expressions.propertyTesters*with a unique *id*.  
  
The *type* reflects to the kind of objects you'd get from the source provider. For now leave it to java.lang.Object.  
  
The identifier of a specific property to check should be a fully qualified name. It consists of a namespace part and a properties name. It is good practice to set the namespace to your Plug-in identifier. Therefore set *namespace* to*com.example.coreexpressions* and add two specific properties *name,validated* to *properties*. Hence our properties are named:

* com.example.coreexpressions.name
* com.example.coreexpressions.validated

Finally implement the class *com.example.coreexpressions.ExamplePropertyTester*:

[view plainprint?](http://codeandme.blogspot.in/2012/04/source-provider-property-tester-and.html)

1. **package** com.example.coreexpressions;
3. **import** org.eclipse.core.expressions.PropertyTester;
5. **public** **class** ExamplePropertyTester **extends** PropertyTester {
7. **private** **static** **final** String NAME = "name";
8. **private** **static** **final** String VALIDATED = "validated";
10. **public** ExamplePropertyTester() {
11. }
13. @Override
14. **public** **boolean** test(Object receiver, String property, Object[] args, Object expectedValue) {
15. **if** (NAME.equals(property))
16. **return** receiver.toString().equals(expectedValue);
18. **if** (VALIDATED.equals(property)) {
19. // do some background checks to see if user is validated
20. **return** **true**;
21. }
23. **return** **false**;
24. }
25. }

The receiver of the *test()* implementation is some object from a source provider. If you set your property tester to only accept objects of a certain type you need to add an *instanceof* check in the test() method.  
  
**Step 3: Your own expression**  
  
To use our source provider and property tester we can either create an expression for some activeWhen, enabledWhen or visibleWhen statement or we can extend org.eclipse.core.expressions.definitions to store the expression for later usage.  
  
Create a new extension for *org.eclipse.core.expressions.definitions* with a unique *id*. Add a *with* section to define which source to use. Set *variable* to the name of our currentUser source: *com.example.coreexpressions.currentUser*.  
  
Add a *test* section to define our custom test. Set *property* to *com.example.coreexpressions.name* and *value* to a string to compare against. In the sample I use "John Doe".  
  
Make sure you activate *forcePluginActivation*. This defines that the Plug-in defining the property tester should be activated. If it is not activated, the property test cannot be performed.  
  
The final definition:

[view plainprint?](http://codeandme.blogspot.in/2012/04/source-provider-property-tester-and.html)

1. **<with**
2. variable="com.example.coreexpressions.currentUser"**>**
3. **<test**
4. forcePluginActivation="true"
5. property="com.example.coreexpressions.name"
6. value="John Doe"**>**
7. **</test>**
8. **</with>**

**Step 4: Updating your source**  
  
Expressions do work right now, but our source is some kind of static. It won't change its status. Open *ExampleSourceProvider*and add following code:

[view plainprint?](http://codeandme.blogspot.in/2012/04/source-provider-property-tester-and.html)

1. **public** **void** setCurrentUser(Object user) {
2. mUser = user;
3. mStatus = (user == **null**) ? "logged off" : "logged on";
5. fireSourceChanged(ISources.WORKBENCH, CURRENT\_USER, mUser);
6. fireSourceChanged(ISources.WORKBENCH, CURRENT\_STATUS, mStatus);
7. }

*fireSourceChanged()* will tell Eclipse that a source changed and therefore forces expressions using this source to be re-evaluated.  
  
To get an instance of your source provider you can query the ISourceProviderService:

[view plainprint?](http://codeandme.blogspot.in/2012/04/source-provider-property-tester-and.html)

1. ISourceProviderService service =(ISourceProviderService) PlatformUI.getWorkbench().getService(ISourceProviderService.**class**);
2. ISourceProvider provider = service.getSourceProvider(ExampleSourceProvider.CURRENT\_USER);

Doc link:

**http://wiki.eclipse.org/Command\_Core\_Expressions#Variables\_and\_the\_Command\_Framework**