CODE & ME

Tuesday, October 9, 2012

Integrating a custom builder

A builder can be used to trigger custom actions during a project build. You can use it to update resource files, generate documentation or to twitter every piece of code you write...

Source code for this tutorial is available on github as a single zip archive, as a Team Project Set or you can browse the files online.

Step 1: Creating the builder

This is the easy part. Create a new *Plug-in project* named *com.codeandme.custombuilder* and switch to the *Extensions* tab of the *plugin.xml*.

Add an extension for *org.eclipse.core.resources.builders*. Set the *ID* to *com.codeandme.custombuilder.myBuilder*, leave the builder settings empty and create a *run* entry below. There set the class to *com.codeandme.custombuilder.MyBuilder*. Implement the class with following code:

```
1  package com.codeandme.custombuilder.builders;
2
3  import java.util.Map;
4
5  import org.eclipse.core.resources.IProject;
6  import org.eclipse.core.resources.IncrementalProjectBuilder;
7  import org.eclipse.core.runtime.CoreException;
8  import org.eclipse.core.runtime.IProgressMonitor;
9
10 public class MyBuilder extends IncrementalProjectBuilder {
11
12  public static final String BUILDER_ID = "com.codeandme.custombuilder.myBuilder";
13
14  @Override
15  protected IProject[] build(final int kind, final Map<String, String> args, final IProgressMoni
16    throws CoreException {
17
18   System.out.println("Custom builder triggered");
19
20   // get the project to build
21   getProject();
22
23   switch (kind) {
24
25   case FULL_BUILD:
26    break;
27
28   case INCREMENTAL_BUILD:
29    break;
30
31   case AUTO_BUILD:
32    break;
33   }
34
35   return null;
36  }
37 }
```

Do not forget to add a plug-in dependency for *org.eclipse.core.runtime*.

Your builder is done. Sure you need to add functionality to it, but this is your part. So now what? We need to add the builder to projects. To selectively add a builder eclipse suggests to use the *Configure* entry in the popup menu of projects.

Step 2: Create context menu entries

First lets create the commands to add and remove our builder.

Add the command definitions to your plugin.xml

```
1 <command
2       defaultHandler="com.codeandme.custombuilder.commands.AddBuilder"
3       id="com.codeandme.custombuilder.addBuilder"
4       name="Add Custom Builder">
5 </command>
6 <command
```

```
      defaultHandler="com.codeandme.custombuilder.commands.RemoveBuilder"
      id="com.codeandme.custombuilder.removeBuilder"
      name="Remove Custom Builder">
</command>
```

At the same time we can add some additional dependencies:

- org.eclipse.core.commands
- org.eclipse.jface
- org.eclipse.ui

Now implement the commands:

```
package com.codeandme.custombuilder.commands;

import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;

import org.eclipse.core.commands.AbstractHandler;
import org.eclipse.core.commands.ExecutionEvent;
import org.eclipse.core.commands.IHandler;
import org.eclipse.core.resources.ICommand;
import org.eclipse.core.resources.IProject;
import org.eclipse.core.resources.IProjectDescription;
import org.eclipse.core.runtime.CoreException;
import org.eclipse.core.runtime.Platform;
import org.eclipse.jface.viewers.ISelection;
import org.eclipse.jface.viewers.IStructuredSelection;
import org.eclipse.ui.handlers.HandlerUtil;

import com.codeandme.custombuilder.builders.MyBuilder;

public class AddBuilder extends AbstractHandler implements IHandler {

 @Override
 public Object execute(final ExecutionEvent event) {
  final IProject project = getProject(event);

  if (project != null) {
   try {
    // verify already registered builders
    if (hasBuilder(project))
     // already enabled
     return null;

    // add builder to project properties
    IProjectDescription description = project.getDescription();
    final ICommand buildCommand = description.newCommand();
    buildCommand.setBuilderName(MyBuilder.BUILDER_ID);

    final List<ICommand> commands = new ArrayList<ICommand>();
    commands.addAll(Arrays.asList(description.getBuildSpec()));
    commands.add(buildCommand);

    description.setBuildSpec(commands.toArray(new ICommand[commands.size()]));
    project.setDescription(description, null);

   } catch (final CoreException e) {
    // TODO could not read/write project description
    e.printStackTrace();
   }
  }

  return null;
 }

 public static IProject getProject(final ExecutionEvent event) {
  final ISelection selection = HandlerUtil.getCurrentSelection(event);
  if (selection instanceof IStructuredSelection) {
   final Object element = ((IStructuredSelection) selection).getFirstElement();

   return (IProject) Platform.getAdapterManager().getAdapter(element, IProject.class);
  }

  return null;
 }

 public static final boolean hasBuilder(final IProject project) {
  try {
   for (final ICommand buildSpec : project.getDescription().getBuildSpec()) {
    if (MyBuilder.BUILDER_ID.equals(buildSpec.getBuilderName()))
     return true;
   }
  } catch (final CoreException e) {
  }

  return false;
 }
}
```

```
package com.codeandme.custombuilder.commands;

import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;

import org.eclipse.core.commands.AbstractHandler;
import org.eclipse.core.commands.ExecutionEvent;
import org.eclipse.core.commands.ExecutionException;
import org.eclipse.core.commands.IHandler;
import org.eclipse.core.resources.ICommand;
import org.eclipse.core.resources.IProject;
import org.eclipse.core.resources.IProjectDescription;
import org.eclipse.core.runtime.CoreException;

import com.codeandme.custombuilder.builders.MyBuilder;

public class RemoveBuilder extends AbstractHandler implements IHandler {

 @Override
 public Object execute(final ExecutionEvent event) throws ExecutionException {
  final IProject project = AddBuilder.getProject(event);

  if (project != null) {
   try {
    final IProjectDescription description = project.getDescription();
    final List<ICommand> commands = new ArrayList<ICommand>();
    commands.addAll(Arrays.asList(description.getBuildSpec()));

    for (final ICommand buildSpec : description.getBuildSpec()) {
     if (MyBuilder.BUILDER_ID.equals(buildSpec.getBuilderName())) {
      // remove builder
      commands.remove(buildSpec);
     }
    }

    description.setBuildSpec(commands.toArray(new ICommand[commands.size()]));
    project.setDescription(description, null);
   } catch (final CoreException e) {
    // TODO could not read/write project description
    e.printStackTrace();
   }
  }

  return null;
 }
}
```

When retrieving the selected project we need to use the *AdapterManager* as some project types do not directly implement *IProject* (that is, if I remember correctly). Then we parse the build specification to add or remove our custom builder.

To add those commands to the *Configure* context menu we create a new menu contribution for *popup:org.eclipse.ui.projectConfigure?after=additions*

```
<extension
      point="org.eclipse.ui.menus">
   <menuContribution
         allPopups="false"
         locationURI="popup:org.eclipse.ui.projectConfigure?after=additions">
      <command
            commandId="com.codeandme.custombuilder.addBuilder"
            style="push">
      </command>
      <command
            commandId="com.codeandme.custombuilder.removeBuilder"
            style="push">
      </command>
   </menuContribution>
</extension>
```

Now you should be able to add and remove your builder.

Step 3: Selectively activate context menu entries

Only one of the commands makes sense regarding the current builder settings of a project. To enrich the user experience we will hide the invalid one.

Therefore we need to use a *PropertyTester* and some *visibleWhen* expressions as we did before in Property testers and Expression examples.

Create a new *propertyTesters* extension.

```
<extension
      point="org.eclipse.core.expressions.propertyTesters">
   <propertyTester
         class="com.codeandme.custombuilder.propertytester.TestBuilderEnabled"
         id="com.codeandme.custombuilder.myBuilderTester"
         namespace="com.codeandme.custombuilder"
         properties="isEnabled"
```

```
        type="java.lang.Object">
    </propertyTester>
</extension>
```

We leave the type to *java.lang.Object* as not all project types use a common base class (except Object of course). Implementing the property tester is straight forward:

```
package com.codeandme.custombuilder.propertytester;

import org.eclipse.core.expressions.PropertyTester;
import org.eclipse.core.resources.IProject;
import org.eclipse.core.runtime.Platform;

import com.codeandme.custombuilder.commands.AddBuilder;

public class TestBuilderEnabled extends PropertyTester {

 private static final String IS_ENABLED = "isEnabled";

 @Override
 public boolean test(final Object receiver, final String property, final Object[] args, final O

  if (IS_ENABLED.equals(property)) {
   final IProject project = (IProject) Platform.getAdapterManager().getAdapter(receiver, IProje

   if (project != null)
    return AddBuilder.hasBuilder(project);
  }

  return false;
 }
}
```

Now add some *visibleWhen* expressions to your menu entries. View the final version of the plugin.xml online.

Posted by Christian Pontesegger at 12:10 PM Labels: Eclipse, RCP

6 comments:

Maarten Meijer December 21, 2012 at 11:42 PM

You don't need a tester for that, as the core expressions gives you access to the project nature, and natures are the only allowed way to link builders to projects:
<visibleWhen
checkEnabled="false">
<with
variable="selection">
<count
value="1">
</count>
<iterate>
<and>
<instanceof
value="org.eclipse.core.resources.IProject">
</instanceof>
<test
property="org.eclipse.core.resources.projectNature"
value="your.nature">
</test>
</and>
</iterate>
</with>
</visibleWhen>

Reply

Anonymous May 7, 2014 at 5:39 AM

hi..
i have a builder and i need to specify that .. how can i do that with this code. here adding and removing of builder is giver.. how can i tell about my builder.

Reply

Replies

Christian Pontesegger May 7, 2014 at 8:06 PM

I do not get your point. This tutorial shows how to register a dedicated builder. Use the MyBuilder.build() method to execute whatever code you need.

This tutorial also shows just one way of attaching a builder. You could also create a new project wizard that creates a dedicated project where your special builder class is registered on project creation

Anonymous May 9, 2014 at 5:42 AM

actually i have compiler. and in eclipse like jdt .. we are running a code as java aplication like that i need to run my code with my compiler. how can i do that

Reply

Anonymous May 7, 2014 at 6:09 AM

if i need to specify a path of my bulder, and i need that builder in that,, how can i do that

Reply

Replies

Christian Pontesegger May 7, 2014 at 8:07 PM

typically you would put a prefs file to a project folder. ".settings/myprefs" is recommended. For configuration you would have to amend project properties though.

Reply

Enter your comment...

Comment as: Unknown (Goo ▾ Sign out

Publish Preview Notify me

Newer Post Home Older Post

Subscribe to: Post Comments (Atom)

Simple theme. Theme images by mariusFM77. Powered by Blogger.