How to extend WindowBuilder 7.0 to support new components.

This is short, may be incomplete description how to add new components to use in WindowlBuilder. Note, that there are at least three levels of support for components:

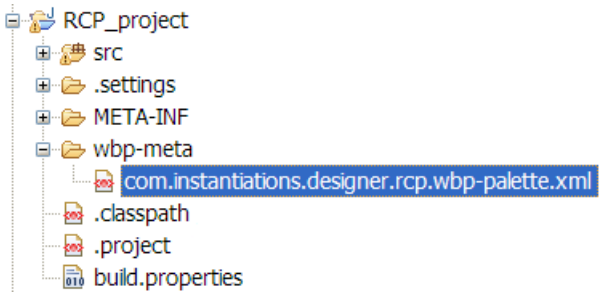
- just add component to palette;
- describe component using *.wbp-component.xml;
- write Java code for special model/layout features.

So, we will look on levels step by step.

1. Add component to palette

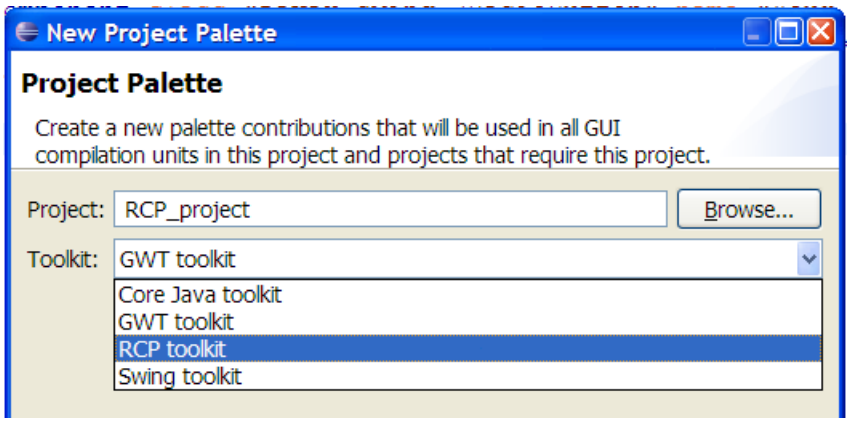
1.1. Palette contribution in project

You can put into project's folder «wbp-meta» file with name toolkitID.wbp-palette.xml with set of «category» and «component» entries.




```
<?xml version="1.0" encoding="UTF-8"?>
<palette>
	<category id="someUniqueId" name="Custom category" description="Category added for project RCP_project"
		open="true">
		<component class="javax.swing.JButton"/>
		<component class="javax.swing.JRadioButton" name="Your name"
			description="You can write any description here."/>
	</category>
</palette>
```

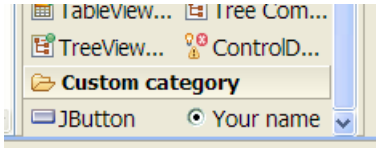
You can use wizard to generate such file for you (note that it always generates Swing components, but this is not important, because this is just example).



Meaning for most attributes in *.wbp-palette.xml is obvious. Attribute `next` specifies id of other

category (in same file or in contribution from plugin.xml) before which this category should be added.

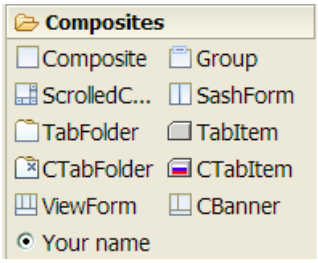
Here is result of previously shown contribution



Note, that you can define `component` outside of new `category`, so put it into some existing standard category, but in this case you should specify `category` attribute.

```
<component category="com.instantiations.designer.rcp.composites"
        class="javax.swing.JRadioButton" name="Your name"/>
```

Icon for component can be places in png or gif format directly near to component class, with same name as component, or in wbp-meta folder, also in same package/folder and same name, but



with .png/.gif extension.

1.2. Palette contribution in jar

In reallity you can put *.wbp-palette.xml file not only in project where you want to edit GUI using WindowBuilder, but also in any project required by your GUI project and even directly in jar with contributed components. So, just by adding my-components.jar into classpath you will also automatically add them to palette. If some project does not inlude jar, you will not see corresponding entries on palette, so they will not consume space.

1.3 Palette contribution from plugin.xml

If you want to use more features, for example specify location of category relative to other categories (use `next` attribute) or show components always and add corresponding jar to classpath automatically, then you should create plugin and use Eclipse extensions to contribute to WindowBuilder palette.

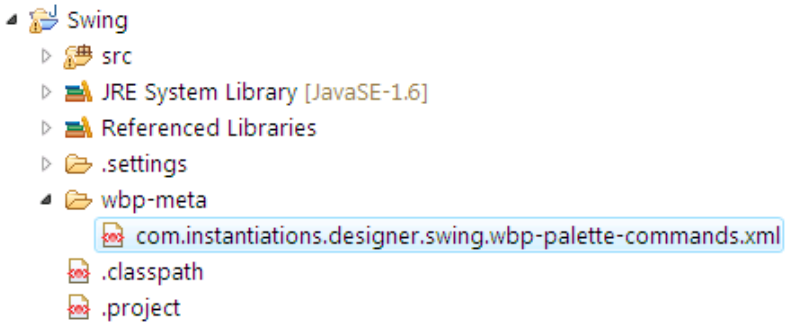
```
<extension point="com.instantiations.designer.core.toolkits">
  <toolkit id="com.instantiations.designer.rcp">
    <classLoader-library bundle="com.instantiations.designer.rcp.nebula.lib" jar="cdatetime-0.9.0.jar"/>
      <palette>
        <category id="com.instantiations.designer.rcp.nebula" name="Nebula" description="Nebula custom widgets"
next="com.instantiations.designer.rcp.FormsAPI">
          <!-- CDateTime -->
          <component class="org.eclipse.swt.nebula.widgets.cdatetime.CButton">
            <library type="org.eclipse.swt.nebula.widgets.cdatetime.CButton"
bundle="com.instantiations.designer.rcp.nebula.lib" jar="cdatetime-0.9.0.jar"/>
          </component>
```

In shown XML `classLoader-library` tell WindowBuilder that this jar should be always added into ClassLoader (even if it is not in classpath), so it would be possible to load components from it and always display on palette. You should also specify `library` element for `component` to tell WindowBuilder that some library should be added to classpath when you first time try to use this component.

For full description of palette related attributes see toolkits.exsd schema.

1.4 Palette commands in project

Sometimes you want to use different palettes for different projects (of same toolkit), or may be you want to share same palette between all developers who work on same project. You can can put into project's folder «wbp-meta» file with name toolkitID.wbp-palette-commands.xml with empty content. When WindowBuilder will find this file, it will save all operations that you perform on palette (move categories and components, add new components, etc) into this file and read from it later.



For example, after moving «Layouts» category before «Containers» this file will have following content:

```
<?xml version="1.0" encoding="UTF-8"?>
<commands>
	<moveCategory
		id="com.instantiations.designer.swing.layouts"
		nextCategory="com.instantiations.designer.swing.containers"/>
</commands>
```

*2. Describe component using *.wbp-component.xml*

WindowBuilder 7.0 allows you describe a lot of information about your component that can be used to provide convient editing. As icon, description for component can be places in directly near to component class, with same name as component, or in wbp-meta folder, also in same package/folder and same name.

Simplest component description look so:

```
<?xml version="1.0" encoding="UTF-8"?>
<component xmlns="http://www.instantiations.com/D2/WBPComponent">
	<model class="com.instantiations.designer.swt.model.widgets.LabelInfo"/>
	<description>Instances of this class represent a non-selectable user interface object that displays a
		string or image. When SEPARATOR is specified, displays a single vertical or horizontal line.</description>
	<!-- CREATION -->
	<creation>
		<source><![CDATA[new org.eclipse.swt.widgets.Label(%parent%, org.eclipse.swt.SWT.NONE)]]></source>
		<invocation signature="setText(java.lang.String)"><![CDATA["New Label"]]></invocation>
	</creation>
	<creation id="separatorHorizontal" name="Horizontal Separator">
		<source><![CDATA[new org.eclipse.swt.widgets.Label(%parent%, org.eclipse.swt.SWT.SEPARATOR |
org.eclipse.swt.SWT.HORIZONTAL)]]></source>
		<description>Horizontal separator.</description>
	</creation>
</component>
```

As you can see, you should specify just description (it also can be skipped, but not recommended, as it is used by default as description for component on palette) and one or more `<creation>` elements.

Note, that plugin com.instantiations.designer.core includes file schema/wbp-component.xsd that is used for validating *.wbp-component.xml files. It sets fairly strong rules on format, so ensure that your XML editor validates description files as you write them.

Components may have different constructors with values bound to some method-based properties. Such constructors rarely used in SWT, but often used in Swing. So, it would be convenient to describe this directly. Here is description for JLabel. As you can see, you can use `property` attribute with method signature (or field name for public field based properties).

```
<!-- CONSTRUCTORS -->
<constructors>
	<constructor>
		<parameter type="java.lang.String" property="setText(java.lang.String)"/>
	</constructor>
	<constructor>
		<parameter type="java.lang.String" property="setText(java.lang.String)"/>
		<parameter type="int" property="setHorizontalAlignment(int)"/>
	</constructor>
	<constructor>
		<parameter type="javax.swing.Icon" property="setIcon(javax.swing.Icon)"/>
	</constructor>
	<constructor>
		<parameter type="javax.swing.Icon" property="setIcon(javax.swing.Icon)"/>
		<parameter type="int" property="setHorizontalAlignment(int)"/>
	</constructor>
	<constructor>
		<parameter type="java.lang.String" property="setText(java.lang.String)"/>
		<parameter type="javax.swing.Icon" property="setIcon(javax.swing.Icon)"/>
		<parameter type="int" property="setHorizontalAlignment(int)"/>
	</constructor>
</constructors>
```

Note, that you don't have to declare all constructors of you component. You need to do this only if you want to say WindowBuilder something useful about constructor, for example that some parameter is bound to some property, or specify default value of parameter (when user presses DEL key on «constructor/parameter» property).

Components may have a lot of properties more than hundred for Swing. But usually you use only handful of them. So, as component developer you should specify which properties should be marked as raraly used (advanced), sometimes used (normal) and often used (preferred). Some properties may be even declared as hidden, so they never will be displayed to user, even if component has it.

```
<!-- PROPERTIES -->
<properties-preferred names="text icon labelFor"/>
<properties-advanced
	names="border disabledIcon displayedMnemonicIndex horizontalTextPosition iconTextGap
verticalTextPosition"/>
<properties-hidden names="UI"/>
```

WindowBuilder understands special tag for property, such as marking in as «text» property, so you can easily edit its text by pressing Space key when component is selected.

```
<property-tag name="text" tag="isText" value="true"/>
```

Properties have different types and for most standard types WindowBuilder includes special property editor to use in properties table. However sometimes standard editor is not good enough, for example for integer enumerations. So, you should specify «configurable property editor».

```
<property id="setVerticalAlignment(int)">
	<editor id="staticField">
		<parameter name="class">javax.swing.SwingConstants</parameter>
		<parameter name="fields">TOP CENTER BOTTOM</parameter>
	</editor>
</property>
<property id="setHorizontalAlignment(int)">
	<editor id="staticField">
		<parameter name="class">javax.swing.SwingConstants</parameter>
		<parameter name="fields">LEFT CENTER RIGHT LEADING TRAILING</parameter>
	</editor>
</property>
```

See Table 1.5 in *.wbp-component.xml documentation for list of existing configurable editors.

If for some reason you don’t want to specify property category using `<properties-*>` element, you can do this separately for some property.

```
<property id="setDisplayedMnemonic(char)">
	<category value="preferred"/>
</property>
```

Some components are not just used separately, but are containers that accept other components. Often description in XML is enough to support this. There are two types of description

based containers: simple (accept only one component) and flow (accept several components, allow to order them). See `simpleContainer` and `flowContainer` documentation.

3. Write Java code

Sometimes descriptions are not enough. For example you need some special editing behavior, so you should write GEF LayoutEditPolicy. In this case you need to write plugin to host your models and policies.

Here is how you can specify model for component in its description:

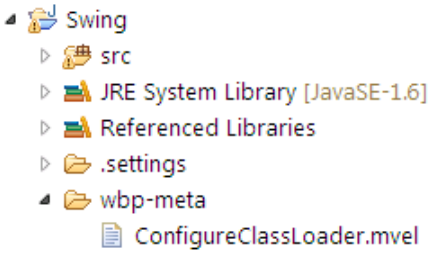
```
<model class="com.instantiations.designer.swing.model.component.ComponentInfo"/>
```

Class ComponentInfo is indirect subclass of JavaInfo that provides basic support for all Java models. Usually all your models will be subclasses of ComponentInfo/ContainerInfo (for Swing) or ControlInfo/CompositeInfo (for SWT).

WindowBuilder provides a lot of broadcast notifications that can be used to be informed about some event, or participate in them. For example you may install listener for adding properties to any component (by your choice), even if it is not yours, see com.instantiations.designer.core.model.broadcast.JavaEventListener.addProperties(JavaInfo, List<Property>). Or you can be informed that some JavaInfo component was deleted, so you may be also want to delete some other resources/source, see com.instantiations.designer.core.model.broadcast.JavaEventListener.deleteAfter(JavaInfo, JavaInfo).

4. Configuring static objects

Sometimes you know that your application configures environment in which it executes. For example loads preferences from file/database, configure default settings of layout managers, etc. In most cases you can not execute same Java configuration code, so WindowBuilder provides support for «design time configuration». To use it create file *`ConfigureClassLoader.mvel`* in folder *`wbp-meta`* of Java project.



MVEL http://mvel.codehaus.org/ is a powerful expression language for Java-based applications. For example this script sets default insets (we don't recommend to use such strange values :-)) for MigLayout on JPanel containers.

```
import net.miginfocom.layout.*;
PlatformDefaults.setPanelInsets(
      new UnitValue(20),
      new UnitValue(50),
      new UnitValue(10),
      new UnitValue(5));
```