Visual Editor New Class Wizard

Change Proposal

Introduction

The Visual Class Wizard is the entry point that allows users to create new Visual Classes in the Visual Editor. However, there is no mechanism for developers to extend the number of styles, or components of those styles, other than altering the Visual Editor (VE) codebase itself. To allow for further extension (the recent effort to add SWT styles to the Visual Editor being a code example), and to better utilize the Eclipse framework and ideology, this proposal attempts to represent an architected means of publicly declaring a method to extend the VE style base.

Current Design: Visual Class Wizard GUI

Figure 1 shows how the New Visual Class Wizard currently represents the Style (Swing, AWT, etc.) and the components of the style (frame, panel, etc.) that the user wants to extend .

Figure 1, The current new visual class wizard

Because the Visual Editor's scope has grown, the current GUI design philosophy has to change to reflect the more expansive role it will undertake. Currently there is an effort to implement SWT support into the Visual Editor. The current GUI does not scale well to encompass the addition of styles or components outside of what the Visual Editor includes with each release.

The scope of the expansion question extends to both the GUI and the underlying code. The use of radio buttons limits the real estate of the available components, relies on there being common components to all styles, and does not allow extensions to styles, or components of those styles, without actually rewriting some GUI code. A more expansive philosophy is required to allow expansion without touching the GUI code, or the underlying "glue code".

Existing Code

In the current design, the styles are statically defined (defined here as widgetSetNames), and retrieved from resource bundles first at the top level:

```
String[] widgetSetNames = new String[] {
        CodegenMessages.getPluginPropertyString("swing") ,
        CodegenMessages.getPluginPropertyString("awt" )
}; //$NON-NLS-1$
```

Then a common set of Visual Elements is defined. In the current model this would apply to both styles.

```
String[] buttonNames = new String[] {
        CodegenMessages.getPluginPropertyString("frame") ,
        CodegenMessages.getPluginPropertyString("panel") ,
        CodegenMessages.getPluginPropertyString("applet")
};
```

For each defined style, a set of superclasses is defined that match the common set of Visual Elements for both styles.

```
String[] swingButtonClasses = new String[] { "javax.swing.JFrame" ,
"javax.swing.JPanel" , "javax.swing.JApplet" };
String[] AWTButtonClasses = new String[] { "java.awt.Frame" , "java.awt.Panel" ,
"java.applet.Applet" };
```

These elements are then propagated to the GUI as shown in Figure 1. If the Swing checkbox button is set to true, the swingButtonClasses are propagated to the Superclass textbox; if set to false, the AWTButtonClasses are propagated to the Superclass textbox. When the user clicks Finish, the high level code path is as follows:

1. The text in superclass field is stored internally. This will form the reference to the newsource extension point, and the basis for selecting the appropriate provider.
2. The contents of the superclass are validated against *swingButtonClasses* and *AWTButtonClasses* to ensure that it is a valid superclass. If not, superclass is set to `java.lang.Object.`
3. The Java Class Creation wizard is allowed to execute and deliver a Compilation Unit of the basic Java Class (modified by the options in the GUI).
4. The contributer for the superclass (shown in the Superclass field in the GUI) is searched for with the method *updateContributor* and stored in: ***private*** *IVisualClassCreationSourceContributor contributor.*
5. The contribute code from the Java jet template is merged in with the Java Class creation wizards compilation unit, using the global merge method: ***protected void*** *merge (ICompilationUnit to, ICompilationUnit from, CodeFormatter formatter, IProgressMonitor monitor)*

Proposed Design: Visual Class Wizard GUI

This section contains the proposal that will be part of Visual Editor 1.0.0

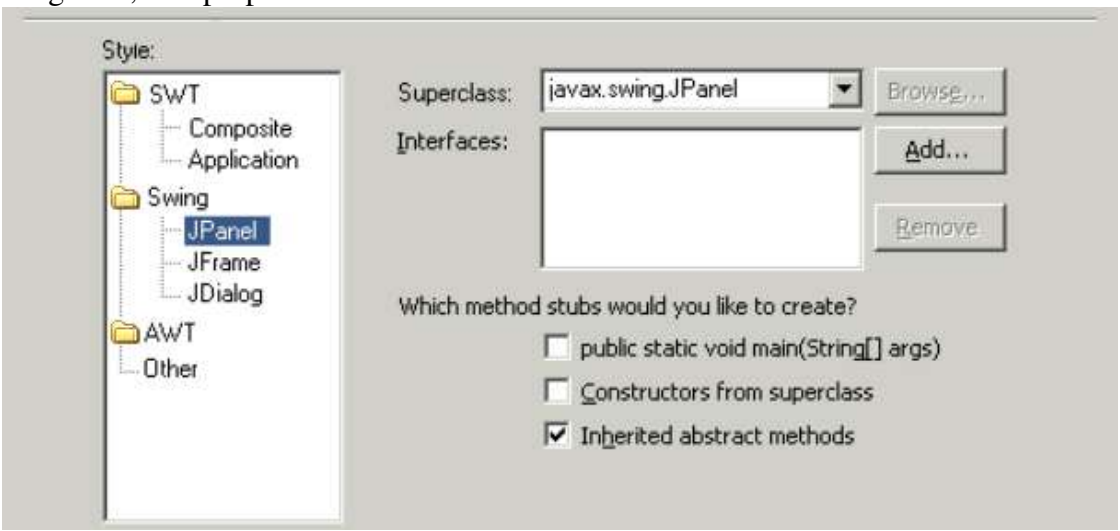
After 0.5.0, work on adding the SWT style to the Visual Editor started (as described in the previous section), it became apparent that the existing GUI design and the initial class-code creation could not lend itself to the different code paradigm that SWT uses.

After discussion on the Eclipse Visual Editor mailing lists (archives available here: https://dev.eclipse.org/mailman/listinfo/ve-dev) the following GUI design was decided upon. This GUI design represents a TreeView widget that replaces the radio buttons and the toggle switch that 0.5.0 defined. A screen shot is available in Figure 2.

The new TreeView widget can be expandable indefinitely and can manage variable-sized lists of styles and components.

The population of the superclass field will remain the same as it is in 0.5.0 (basically every time you click a component of a style, the corresponding superclass is populated to the Superclass field).

Figure 2, The proposed new visual class wizard



Beyond the visual changes to the widgets on the new Visual Class Creation Wizard, there are several other areas that will change.

Style/Component Extension Point

All styles and their components (including the existing Swing/AWT) will now be populated through a new extension point. This extension point absorbs the existing "newsource" extension point (for code contributions), and also encapsulates the elements needed to define styles, and components. In Appendix A there is an attached proposed schema for the extension point. For purposes of illustration, below is an example of how an extension will contribute to the new extension point.

```
<extension
    point="org.eclipse.ve.java.core.newStyleComponet">
    <category
        name="Swing"
        id="org.eclipse.ve.java.core.swing">
    </category>
    <visualElement
        name="Frame"
        category=""org.eclipse.ve.java.core.swing"
        type="javax.swing.JFrame
        contributor="org.eclipse.ve.internal.java.codegen.wizards.contributors.ContentPaneSourceContributor">
    </visualElement>
  <visualElement
        name="Panel"
        category=""org.eclipse.ve.java.core.swing"
        type="javax.swing.JPanel"
        contributor=""org.eclipse.ve.internal.java.codegen.wizards.contributors.ComponentSetSizeSourceContributor">
    </visualElement>
        <visualElementname="Applet"
        category=""org.eclipse.ve.java.core.swing"
        type="javax.swing.JApplet"
        contributor=""org.eclipse.ve.internal.java.codegen.wizards.contributors.ContentPaneSourceContributor">
    </visualElement>
```

Style/Component Interface

The interface is rudimentary--it has only one method: *contributeSource*, which takes the Java Class created by the Java Class Wizard, and merges/overrides/add the contributing source with the Compilation Unit passed to it.

```
public interface IVisualClassCreationSourceGenerator {

	public static final String CREATE_MAIN = "createMain";
	public static final String CREATE_SUPER_CONSTRUCTORS =
	"createSuperConstructors";
	public static final String CREATE_INHERITED_ABSTRACT =
	"createInheritedAbstract";

	/**
	* Contribute code and merge with given Java Class
	Compilation Unit
	*
	*/
	public ICompilationUnit contributeSource(ICompilationUnit
	from, CodeFormatter formatter, HashMap argumentMatrix);
}
```

Argument Description:

ICompilationUnit from – this is the compilation unit generated by the superclass New Java Class.

CodeFormatter formatter – Applicable formatter

HashMap argumentMatrix – this is a HashMap containing the method stub argument values defined in the GUI. The keys are as follows:

- *`createMain`* – Create the method stub **public static void main(String[] args)**
- *`createSuperConstructors`* – Create constructors from the superclasses
- *`createInheritedAbstract`* – Created inherited abstract methods

The implementation, and resolving if the argument is applicable in that contributors given case, is the responsibility of the contributor for that superclass.

Most of the reconciliation between the generic Java class has been shifted from the Visual Class Creation Wizard to the contributor. In moving away from Java Jet (and thus a template-based design) for this section of the wizard, the means to merge code in a generic manner is no longer possible. Therefore we ask the contributors to merge their code with the Java class compilation unit, as each contributor can deal with the individual implementation details in an abstract and independent manner.

Code Changes

With the new new extension point in place, there will be several areas of code changes in place.

General Changes

- Remove all static references in the code relating to styles, resource bundles, radio buttons, and logic associated with the old model.
- Add the treeview GUI component to the wizard. Define a method that will collate all styles and components contributed, and populates the treeview.
- Define a selection event that populates the Superclass text field on selection events in the treeview.

The largest area of change is in the methodology for contributing code. With the interface defined above, the contributor is now responsible for contributing the relevant code stub, merging the compilation unit produced by the class creation wizard, and providing the finished product back to the Visual Class Creation Wizard.

A high level overview is as follows:

1. Collate all the extension points, and populate the treeview from the contributed styles and components.
2. Discard and log any components that have a style id that does not exist.
3. Wait until selections are finalized and the Finish button is clicked
4. Allow Java Class Creation wizard to create the basic type.
5. Instantiate the relevant contributing extension.
6. Call *contributeSource* from the contributing extension, providing the Java Class compilation unit, the wizard argument matrix.

Extension Point Schema

Visual Class - New Style Component

Identifier:

org.eclipse.ve.java.core.newStyleComponent

Since:

May 11th 2004

Description:

This extension point is used to register new visual class styles and component extensions. New styles and components appear as choices within the "Style" treeview of the Visual Class Creation Wizard, and are typically used to create new style classes for the Visual Editor.

Configuration Markup:

<!ELEMENT extension EMPTY>

<!ATTLIST extension

point CDATA #REQUIRED>

<!ELEMENT category EMPTY>

<!ATTLIST category

name CDATA #IMPLIED

id CDATA #IMPLIED>

- **name** -
- **id** -
-

<!ELEMENT visualElement EMPTY>

<!ATTLIST visualElement

name CDATA #REQUIRED

category CDATA #REQUIRED

type CDATA #REQUIRED

contributor CDATA #IMPLIED>

- **name** -
- **category** -
- **type** -
- **contributor** -

Examples:

Following is an example of creation wizard configuration:

```
<extension
      point="org.eclipse.ve.java.core.newStyleComponent">

      <category
            name="Swing"
            id="org.eclipse.ve.java.core.swing">
      </category>

      <visualElement
            name="Frame"
            category="org.eclipse.ve.java.core.swing"
            type="javax.swing.JFrame”
            contributor="org.eclipse.ve.internal.java.codegen.wizards.contributors.ContentPaneSourceContributor">

      </visualElement>
</extension>
```

API Information:

The value of the class attribute in visualElement must represent a class that implements org.eclipse.ve.java.core.IVisualClassCreationSourceGenerator

Document Changelog

May 12th – Phil Muldoon, Red Hat, Draft released

May 12th – Mike Behm, Red Hat, issues corrections to the XML code (not correctly terminated tags), and general editorial suggestions.

May 12th – Peter Walker, IBM, proposes that a mechanism be defined to pass all of the GUI arguments to the contributor (denoted as *Which method stubs would you like to create?*). Discussion follows that a Property object is too heavyweight. Settle on HashMap as the associative method to pass arguments to the contributor.

May 12th – Dr Gili Mendel, IBM, proposes to make changes to the GUI to save screen real estate. Included proposed screenshot

May 12th – Dr Gili Mendel, IBM, comments on the multiple constructors for each style. Implementation details need further investigation. A side product of that discussion resulted in the interface being collapsed to one method: *contributeSource* that now replaces the two methods: *generateSource* and *mergeSource*

May 14th – Dr Gili Mendel, IBM, suggests we enumerate the HashMap arguments constants into the interface.