DeveloperManual JMeterGuiBasics

Basics of Making JMeter Gui Element

When writing any Jmeter component, there are certain contracts you must be aware of – ways a Jmeter component is expected to behave if it will run properly in the Jmeter environment. This section describes the contract that the GUI part of your component must fulfill.

GUI code in Jmeter is strictly separated from Test Element code. Therefore, when you write a component, there will be a class for the Test Element, and another for the GUI presentation. The GUI presentation class is stateless in the sense that it should never hang onto a reference to the Test Element (there are exceptions to this though).

A gui element should extend the appropriate abstract class provided:

- AbstractSamplerGui
- AbstractAssertionGui
- AbstractConfigGui
- AbstractControllerGui
- AbstractPostProcessorGui
- AbstractPreProcessorGui
- AbstractVisualizer
- AbstractTimerGui

These abstract classes provide so much plumbing work for you that not extending them, and instead implementing the interfaces directly is hardly an option. If you have some burning need to not extend these classes, then you can join me in IRC where I can convince you otherwise 🙂.

So, you've extended the appropriate gui class, what's left to do? Follow these steps:

1. Implement getResourceLabel()
    a. This method should return the name of the resource that represents the title/name of the component. The resource will have to be entered into Jmeters messages.properties file (and possibly translations as well).
2. Create your gui. Whatever style you like, layout your gui. Your class ultimately extends Jpanel, so your layout must be in your class's own Content Pane. Do not hook up gui elements to your TestElement class via actions and events. Let swing's internal model hang onto all the data as much as you possibly can.
    a. Some standard gui stuff should be added to all Jmeter gui components:
        i. call setBorder(makeBorder()) for your class. This will give it the standard Jmeter border
        ii. add the title pane via makeTitlePanel(). Usually this is the first thing added to your gui, and should be done in a Box vertical layout scheme, or with Jmeter's VerticalLayout class. Here is an example init() method:

```
private void init()
{
    setLayout(new BorderLayout());
    setBorder(makeBorder());

    Box box = Box.createVerticalBox();
    box.add(makeTitlePanel());
    box.add(makeSourcePanel());
    add(box,BorderLayout.NORTH);
    add(makeParameterPanel(),BorderLayout.CENTER);
}
```

1.#3 Implement public void configure(TestElement el)

    b. Be sure to call super.configure(e). This will populate some of the data for you, like the name of the element.
    c. Use this method to set data into your gui elements. Example:

```
public void configure(TestElement el)
{
    super.configure(el);
    useHeaders.setSelected(el.getPropertyAsBoolean(RegexExtractor.USEHEADERS));
    useBody.setSelected(!el.getPropertyAsBoolean(RegexExtractor.USEHEADERS));
    regexField.setText(el.getPropertyAsString(RegexExtractor.REGEX));
    templateField.setText(el.getPropertyAsString(RegexExtractor.TEMPLATE));
    defaultField.setText(el.getPropertyAsString(RegexExtractor.DEFAULT));
    matchNumberField.setText(el.getPropertyAsString(RegexExtractor.MATCH_NUM));
    refNameField.setText(el.getPropertyAsString(RegexExtractor.REFNAME));
}
```

1.#4 implement public void modifyTestElement(TestElement e). This is where you move the data from your gui elements to the TestElement. It is the logical reverse of the previous method.

    d. Call super.configureTestElement(e). This will take care of some default data for you.
    e. Example:

```
public void modifyTestElement(TestElement e)
{
    super.configureTestElement(e);
    e.setProperty(new BooleanProperty(RegexExtractor.USEHEADERS,useHeaders.isSelected()));
    e.setProperty(RegexExtractor.MATCH_NUMBER,matchNumberField.getText());
    if(e instanceof RegexExtractor)
    {
        RegexExtractor regex = (RegexExtractor)e;
        regex.setRefName(refNameField.getText());
        regex.setRegex(regexField.getText());
        regex.setTemplate(templateField.getText());
        regex.setDefaultValue(defaultField.getText());
    }
}
```

1.#5 implement public TestElement createTestElement(). This method should create a new instance of your TestElement class, and then pass it to the modifyTestElement(TestElement) method you made above.

```
public TestElement createTestElement()
{
    RegexExtractor extractor = new RegexExtractor();
    modifyTestElement(extractor);
    return extractor;
}
```

The reason you cannot hold onto a reference for your Test Element is because Jmeter reuses instance of gui class objects for multiple Test Elements. This saves a lot of memory. It also makes it incredibly easy to write the gui part of your new component. You still have to struggle with the layout in Swing, but you don't have to worry about creating the right events and actions for getting the data from the gui elements into the TestElement where it can do some good. Jmeter knows when to call your configure, and modifyTestElement methods where you can do it in a very straightforward way.