DeveloperManual TestBeanTutorial

Tutorial - Making a Test Bean

In this part, we will go through the process of creating a simple component for Jmeter that uses the new TestBean framework. This component will be a CSV file reading element that will let users easily vary their input data using csv files. To most effectively use this tutorial, open the three files specified below (found in Jmeter's src/components directory).

1. Pick a package and make three files:
    - [ComponentName].java (org.apache.jmeter.config.CSVDataSet.java)
    - [ComponentName]BeanInfo.java (org.apache.jmeter.config.CSVDataSetBeanInfo.java)
    - [ComponentName]Resources.properties (org.apache.jmeter.config.CSVDataSetResources.properties)
2. CSVDataSet.java must implement the TestBean interface. In addition, it will extend ConfigTestElement, and implement LoopIterationListener.
    - TestBean is a marker interface, so there are no methods to implement.
    - Extending ConfigTestElement will make our component a Config element in a test plan. By extending different abstract classes, you can control the type of element your component will be (ie AbstractSampler, AbstractVisualizer, GenericController, etc - though you can also make different types of elements just by instantiating the right interfaces, the abstract classes can make your life easier).
3. CSVDataSetBeanInfo.java should extend org.apache.jmeter.testbeans.BeanInfoSupport create a zero-parameter constructor in which we call super(CSVDataSet.class); we'll come back to this.
4. CSVDataSetResources.properties - blank for now
5. Implement your special logic for you plugin class.
    a. The CSVDataSet will read a single CSV file and will store the values it finds into JMeter's running context. The user will define the file, define the variable names for each "column". The CSVDataSet will open the file when the test starts, and close it when the test ends (thus we implement TestListener). The CSVDataSet will update the contents of the variables for every test thread, and for each iteration through its parent controller, by reading new lines in the file. When we reach the end of the file, we'll start again at the beginning. When implementing a TestBean, pay careful attention to your properties. These properties will become the basis of a gui form by which users will configure the CSVDataSet element.
    b. Your element will be cloned by JMeter when the test starts. Each thread will get it's own instance. However, you will have a chance to control how the cloning is done, if you need it.
    c. Properties: *filename*, *variableNames*. With public getters and setters.
        - **filename** is self-explanatory, it will hold the name of the CSV file we'll read
        - **variableNames** is a String which will allow a user to enter the names of the variables we'll assign values to. Why a String? Why not a Collection? Surely users will need to enter multiple (and unknown number of)variable names? True, but if we used a List or Collection, we'd have to write a gui component to handle collections, and I just want to do this quickly. Instead, we'll let users input comma-delimited list of variable names.
    d. I then implemented the IterationStart method of the LoopIterationListener interface. The point of this "event" is that your component is notified of when the test has entered it's parent controller. For our purposes, every time the CSVDataSet's parent controller is entered, we will read a new line of the data file and set the variables. Thus, for a regular controller, each loop through the test will result in a new set of values being read. For a loop controller, each iteration will do likewise. Every test thread will get different values as well.
6. Setting up your gui elements in CSVDataSetBeanInfo:
    - You can create groupings for your component's properties. Each grouping you create needs a label and a list of property names to include in that grouping. Ie:

```
createPropertyGroup("csv_data",new String[]{"filename","variableNames"});
```

    - Creates a grouping called "csv_data" that will include gui input elements for the "filename" and "variableNames" properties of CSVDataSet. Then, we need to define what kind of properties we want these to be:

```
p = property("filename");
p.setValue(NOT_UNDEFINED, Boolean.TRUE);
p.setValue(DEFAULT, "");
p.setValue(NOT_EXPRESSION,Boolean.TRUE);
p = property("variableNames");
p.setValue(NOT_UNDEFINED, Boolean.TRUE);
p.setValue(DEFAULT, "");
p.setValue(NOT_EXPRESSION,Boolean.TRUE);
```

This essentially creates two properties whose value is not allowed to be null, and whose default values are "". There are several such attributes that can be set for each property. Here is a rundown:

Key Value	Description of Value
NOT_UNDEFINED	The property will not be left null if true.
DEFAULT	A default values must be given if NOT_UNDEFINED is true.
NOT_EXPRESSION	The value will not be parsed for functions if this is true.
NOT_OTHER	This is not a free form entry field – a list of values has to be provided

<ac:structured-macro ac:name="unmigrated-wiki-markup" ac:schema-version="1" ac:macro-id="b21c3ad6-cc25-43f3-a7a3-de1aa4e6d82a" ><ac:plain-text-body><![CDATA[	TAGS	With a String[] as the value, this sets up a predefined list of acceptable values, and JMeter will create a dropdown select.	]]></ac:plain-text-body></ac:structured-macro>
TableEditor.CLASSNAME	The name of the Class that will represent each row of a GUI Table (only valid if you have set the property's editor class to **TableEditor** via `p.setPropertyEditorClass(TableEditor.class);`		
<ac:structured-macro ac:name="unmigrated-wiki-markup" ac:schema-version="1" ac:macro-id="e81a988f-724d-4315-a902-a96f413fb395" ><ac:plain-text-body><![CDATA[	[TableEditor].HEADERS	A String[] array that holds the header labels for the table, if using [TableEditor]	]]></ac:plain-text-body></ac:structured-macro>
<ac:structured-macro ac:name="unmigrated-wiki-markup" ac:schema-version="1" ac:macro-id="4ab12a63-0b86-46b2-8602-b799f5c3cc51" ><ac:plain-text-body><![CDATA[	[TableEditor].OBJECT_PROPERTIES	A String[] array that holds the names of the properties you wish to display (and make editable) for each object in the table (ie these will correspond to the columns of the table). If the CLASSNAME is "java.lang.String" then this should be skipped	]]></ac:plain-text-body></ac:structured-macro>
MULTILINE	If true, then the component for the property will be allowed to grow with the screen space in the window - ideal for TestArea editors, scroll panels, and other more complicated gui elements		

Additionally, a custom property editor can be specified for a property:

```
p.setPropertyEditorClass(FileEditor.class);
```

This will create a text input plus browse button that opens a dialog for finding a file. Usually, complex property settings are not needed, as now. For a more complex example, look at org.apache.jmeter.protocol.http.sampler.AccessLogSamplerBeanInfo

1.#7 Defining your resource strings. In **CSVDataSetResources.properties** we have to define all our string resources. To provide translations, one would create additional files such as CSVDataSetResources_ja.properties, and CSVDataSetResources_de.properties. For our component, we must define the following resources:

- **displayName** - This will provide a name for the element that will appear in menus.
- **csv_data.displayName** - we create a property grouping called "csv_data", so we have to provide a label for the grouping
- **filename.displayName** - a label for the filename input element.
- **filename.shortDescription** - a tool-tip-like help text blurb.
- **variableNames.displayName** - a label for the variable name input element.
- **variableNames.shortDescription** - tool tip for the variableNames input element.

1.#8 Debug your component.

Remark on variable naming and properties:

- If you have got a field called "aVariable", the getter and setter should be named like getAVariable/setAVariable (so far that is standard)
- In the .properties file you have to refer to this variable by "AVariable" (capital "A"!).
- The same in the BeanInfoSupport Class: Refer to the variable with "AVariable".

Troubleshooting

- If the constructor for your BeanInfo class is not public, your property settings will not take effect in JMeter. Ensure the constructor is public.