JMeterTestElements

JMeter defines the following types of test elements:

Controllers

Controllers contain other test elements – most significantly Samplers. Controllers implement the Controller interface, whose most relevant method is next(), which returns the next Sampler to be used.

Q: why is this done in this way, instead of handling control over to a controller run() method?

- **Answer:** The idea was to mimic the Iterator interface and style of functioning.

Threads

Q: Is a Thread Group a controller?

- **Answer:** essentially yes, since it implements the Controller interface. It implements the functionality by containing a LoopController

Q: What's the relationship between a Thread Group and the JMeterThread objects?

- **Answer:** Not much at all. JMeterThread is the JMeter engine itself that runs the test. A ThreadGroup is the top level of the test where the JMeterThread begins the process of iterating through the test tree.

Q: Does this mean that a Thread Group is just a Loop Controller that can be placed at the top of the tree?

- **Answer:** Sort of. The problem is that a test can have multiple ThreadGroups, so it is not the "top" of the tree. The JMeter Engine searches the tree for ThreadGroup objects. Each one it finds results in threads spawning. The ThreadGroup also provides additional information about that ThreadGroup (ie number of threads, delay between each).

Samplers

Samplers take measurements (usually by generating some load and measuring how long it takes). Their most relevant method is the sample() method, which returns a SampleResult. Samplers are always cloned before being used – which means they can change as they go, but also causes performance concerns.

Q: What/who takes care of associating configuration elements and modifiers to the Samplers?

- **Answer:** The Samplers. Do you mean in terms of the menu options available when a user right-clicks on the Sampler? The Sampler is responsible, but most samplers, and most TestElements just use the default behavior provided by the abstract classes provided. If you mean in terms of merging elements at runtime, then, again, the Sampler is responsible. All TestElements are responsible for knowing how to merge with any other TestElement, via the addTestElement() method. In most Samplers, you can see this at work in their addCustomTestElement() method. The default behavior tends to be not to do any merging.

Q: And who takes care of calling the addTestElement() method for each of the applicable configuration elements and modifiers? Is this done at compile-time or at run-time?

- **Answer:** org.apache.jmeter.threads.TestCompiler. This is done at run-time.

Timers

...(to be filled)...

Q: How does the Thread learn about the Timers applicable to a certain Sampler?

- **Answer:** In the org.apache.jmeter.threads package is a class called TestCompiler. It implements the HashTreeTraverser interface and is responsible for traversing the test tree and extracting all the information it needs to know things like that. Samplers are gathered into Sampler Packages that contain all the TestElements associated with them (determined by tree hierarchy). This includes Timers, Modifiers, Config Elements, Assertions, Listeners, etc).

Q: Are these Sampler Packages created at compile-time and then cycled through in the right order? Or they are created at run-time as they are needed?

- **Answer:** The packages are actually created before the run (so, compile-time is the answer). They are maintained in a HashMap with the Sampler as the key for each of it's packages. When the test iteration calls for a sampler, the TestCompiler checks this map and delivers the whole package - after doing some processing on it (ie resolving functions, merging elements).

Listeners / Visualizers

Config Elements

Assertions

Modifiers and Response-Based Modifiers

.....