JMeterVariablesAndFunctions

JMeter's TestElements are essentially specialized HashMap objects. For most elements, the JMeter engine can traverse through a Test Element and parse all values within, looking for functions and/or user-defined variables. At the present, this is not perfect, so there are times when a function or variable in a Test Element will not get processed, but for the most part, any function or variable appearing anywhere within any Test Element will be processed.

Top-level user-defined variables (ie those the user has defined in the TestPlan gui object) are found and replaced at compile time, before the test actually starts running.

Functions and variables not found are essentially "marked" for processing at runtime. Essentially, any string value that the parser determines contains functions and/or unknown variables is replaced in it's entirety by a CompoundFunction object. The CompoundFunction object is also responsible for parsing these individual strings, looking for functions and variables.

Currently, the CompoundFunction object performs a complex two-pass operation on each string. I believe this would be better off with a fully recursive algorithm that allowed more flexibility (ie functions within functions as parameters, etc).

At runtime, any TestElement flagged as "containing functions" is processed and the functions and unknown variables are resolved. This is done in the org.apache.jmeter.threads.TestCompiler class. This is a simple procedure, as each function is simply asked to resolve itself. Functions and implementations of org.apache.jmeter.testelement.ThreadListener are provided access to a class called org.apache.jmeter.threads.JMeterVariables, which is a holding area for each thread to store variable values and other information during test run.

Simply by implementing the ThreadListener interface, any class can participate in storing and retrieving variables and their values. For this reason, making an new Extractor object type should be easy - any Extractor will implement ThreadListener and store extracted values therein. These values will automatically be inserted any time a user references the variable (ie ${varName}). Of course, this assumes the Extractor is told what variable name to store it's info in (obviously, has to match the previous "varName").