# BuildingPlugInWithEclipse

## Building JMeter plug-in with Eclipse

Eclipse is a very popular Java development environment.

Targetted to Eclipse 3.1 and JMeter 2.1 trunk version. JMeter isn't designed to be edited and build from Eclipse IDE (JMeter uses Ant build file and complex path/jar structure with optional dependencies), so some work has to be done before we get wheels running.

**NOTE: This page is very out of date**

## Building JMeter from sources with Eclipse

It's useful to have JMeter Eclipse project in hand, since you need to do debugging and possible bug fixing in JMeter itself.

1. Check out JMeter trunk from Subversion
    - Get Subclipse plug-in for Eclipse http://subclipse.tigris.org/
    - Check out http://svn.apache.org/repos/asf/jmeter/rel-2-2 as jmeter-trunk Java project
    - Trunk might not be the latest version. Ask from JMeter mailing list which is the active development branch of JMeter.

2. Set up jmeter-trunk project

- Let Eclipse internal compiler handle the building of JMeter, instead of relying on JMeter's own Ant build files. When Eclipse handles building, development work goes more smoothly.
- Source build paths:
    - Add all paths which contain /org folder
        - src/monitor and src/protocol contain nested levels so be careful choosing the right target folders
        - src/protocol/jndi has a strange directory structure so skip this path
    - Exclude following files from building (they have missing dependencies)
        - org/apache/jmeter/util/JsseSSLManager.java (HTTP security)
        - org/apache/jmeter/util/keystore/DefaultKeyStore.java (HTTP security)
        - org/apache/jmeter/util/keystore/PKCS12KeyStore.java (HTTP security)
        - org/apache/jmeter/protocol/http/sampler/WebServiceSampler (depend on javax.mail)
        - org/apache/jmeter/protocol/http/control/gui/WebServiceSamplerGui (depend on javax.mail)
        - org/apache/jmeter/reporters/ (depend on javax.mail)
    - ...or just copy missing JARs from Internet (Java MAIL API: mailapi.jar, etc.), it's easier
    - ...or just add mailapi.jar from Internet to build path and remove src/protocol/jms and src/htmlparser16 from source folders
    - Add libraries
        - All jar files under lib/ folder
    - Export libraries
        - At least logkit libraries are needed in projects depended on JMeter
        - Export them at order and export tab page in the project properties dialog
    - To make log4j logging to Eclipse console work, you need to change some code. You can find the affected code in the org.apacha.jmeter.jorphan.logging.LoggingManager.initializeLogging(Properties) method. Read more notes about this below.

```
public static void initializeLogging(Properties properties) {
    if (logManager == null) {
        logManager = new LoggingManager();
    }

    setFormat(properties);

    setPriority(properties.getProperty(LOG_PRIORITY, "INFO"));

    // Direct to system out

    isWriterSystemOut = true;
    isTargetSystemOut = true;

    WriterTarget wt = new WriterTarget(new OutputStreamWriter(System.out), getFormat());
    Hierarchy.getDefaultHierarchy().setDefaultLogTarget(wt);

    setLoggingLevels(properties);

    // if (logManager == null) {
    // logManager = new LoggingManager();
    // }
    //
    // setFormat(properties);
    //
    // // Set the top-level defaults
    // setTarget(makeWriter(properties.getProperty(LOG_FILE, "jmeter.log"), LOG_FILE));
    // setPriority(properties.getProperty(LOG_PRIORITY, "INFO"));
    //
    // setLoggingLevels(properties);
    // // now set the individual categories (if any)
    //
    // setConfig(properties);// Further configuration
}
```

- Now Eclipse should build JMeter without errors (stop icons in source tree)

## Setting up plug-in project

This Eclipse project will build your custom components and launch JMeter with a configuration which will find them classes propeply.

1. Set up your custom plug-in project
    - Create a new Java project
    - Add jmeter-trunk to project dependencies
    - Create bin/ and src/ folders
    - Create the *src/jmeter.properties* file
        - This defines where JMeter internal class loader looks for classes
        - It should look like this (please note that you have to adapt the first part of the search_paths property so it points to your JMeter Eclipse project!)

```
# JMeter uses case sensitive string matching to test these paths against absolute class path.
# This is very bad since paths must be *exactly* here as they appear in Java classpath.
# Hopefully this will be fixed in the future versions. Note Windows needs to escape \ character as \\.

search_paths=C:\\icecom\\jmeter-2.2\\bin;.

# File that holds a record of name changes for backward compatibility issues
upgrade_properties=upgrade.properties

# Should JMeter automatically load additional JMeter properties?
# File name to look for (comment to disable)
user.properties=user.properties

# Should JMeter automatically load additional system properties?
# File name to look for (comment to disable)
system.properties=system.properties

log_level.jorphan=WARN

log_level.jmeter.testbeans=WARN

log_level.jmeter.gui=WARN

log_level.jmeter.engine.util=WARN

log_level.jmeter.threads.util=WARN

log_level.fi.xxx=TRACE

# Widen default log output for Eclipse console
log_format=%-6.6{priority} (%-10.10{thread}) [%-25.25{category}] %{message}
                          .
```

- Create the *src/log4j.conf* file

```
log4j.rootLogger=debug, stdout

log4j.appender.stdout=org.apache.log4j.ConsoleAppender
log4j.appender.stdout.layout=org.apache.log4j.PatternLayout

# Pattern to output the caller's file name and line number.
log4j.appender.stdout.layout.ConversionPattern=%5p [%t] (%F:%L) - %m%n
```

2. Setting up a JMeter launcher

- Set working directory to bin/ in your project
  - JMeter is hard coded to look up *jmeter.properties* in the launch folder
  - E.g. ${workspace_loc:SIPPerformanceStresser/bin}
- Copy lib/ from jmeter-trunk to your plug-in project root
  - JMeter is hard coded to look up jars from ../lib
- Main class: org.apache.jmeter.NewDriver
-

## Creating components

JMeter 2.x uses its internal TestBean-framework for adding new components. For more information, see Tutorial - Making a JMeter TestBean, this PDF and JMeter source code

For each component you need

1. An element class which interits from [TestElement, ConfigurationElement, XXXElement] and implements [TestBean] interface. [TestBean] interface marks classes which JMeter plug-in class loader loads automatically.

2. Each TestBean class needs BeanInfoSupport class whichs describes the properties of the element class. This class name **must be** MyTestElement + BeanInfo, e.g. MyTestElementBeanInfo, or the class loader doesn't find it.

3. Properties file which gives out user interface strings for properties. The name of the file **must be** MyTestElement + Resources.properties, e.g. MyTestElementResources.properties.

JMeter framework doesn't report about missing files, so be careful with the filenames.

## Example

Here are some examples with some dummy non-working implemenation details.

Java source fi/xxx/jmeter/sip/core/SIPReceiver

```
package fi.xxx.jmeter.sip.core;

import org.apache.jmeter.samplers.AbstractSampler;
import org.apache.jmeter.samplers.Entry;
import org.apache.jmeter.samplers.SampleResult;
import org.apache.jmeter.testbeans.TestBean;

public class SIPReceiver extends AbstractSampler implements TestBean {

        public SampleResult sample(Entry e) {
                return null;
        }

}
```

Java source fi/xxx/jmeter/sip/core/SIPReceiverBeanInfo

```
package fi.xxx.jmeter.sip.core;

import java.beans.PropertyDescriptor;

import org.apache.jmeter.config.CSVDataSet;
import org.apache.jmeter.testbeans.BeanInfoSupport;

public class SIPReceiverBeanInfo extends BeanInfoSupport {

        public SIPReceiverBeanInfo() {
                super(SIPReceiver.class);

                createPropertyGroup("sip_receiver", new String[] { "filename", "variableNames", "delimiter" });
                PropertyDescriptor 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);
                p = property("delimiter");
                p.setValue(NOT_UNDEFINED, Boolean.TRUE);
                p.setValue(DEFAULT, ",");
                p.setValue(NOT_EXPRESSION, Boolean.TRUE);
        }
}
```

Properties file fi/xxx/jmeter/sip/core/SIPreceiverResources.properties

```
displayName=SIP Receiver
sip_receiver.displayName=Configure SIP receiver
filename.displayName=Filename
filename.shortDescription=Name of the file (within your supporting file directory) that holds cvs data
variableNames.displayName=Variable Names (comma-delimited)
variableNames.shortDescription=List your variable names in order to match the order of columns in your csv
data.  Separate by commas.
delimiter.displayName=Delimiter (use '\\t' for tab)
delimiter.shortDescription=Enter the delimiter ('\\t' for tab)
```

## Running from command line

When developing your test component, you need to modify your source code often. This leads to restarting the JMeter. JMeter launch tooks a while, killing your productivity.

If you are not developing UI related functionality, create a test plan, save it, and start JMeter from command line without GUI. Put following to your program arguments:

--nongui --testfile C:\zzz\SIPPerformanceStresser\bin\siptest.jmx

## Notes

### Faulty jorphan logging launcher

I wasted 4 hours trying to get Jorphan (Jmeter component) logging to work. Jorphan overrides log4j normal configuration mechanism, but does it badly, making usage of custom log4j settings impossible. Specifically initializeLogging always overrides previous settings with its own default settings.

1. There is one missing dependency which causes start-up failure with message Caused by: java.lang.NoClassDefFoundError: org/apache/avalon/excalibur/i18n/ResourceManager if you try to configure logger via jmeter.properties. Get the jar file from http://mirrors.bevc.net/apache/excalibur/excalibur-i18n/binaries/ and place it to lib/

2. Even after placing a correct logging file through *jmeter.properties* settings you get this

```
org.apache.avalon.framework.configuration.ConfigurationException: No log targets configured for the root logger.
        at org.apache.avalon.excalibur.logger.LogKitLoggerManager.setupLoggers(LogKitLoggerManager.java:531)
        at org.apache.avalon.excalibur.logger.LogKitLoggerManager.configure(LogKitLoggerManager.java:407)
        at org.apache.jorphan.logging.LoggingManager.setConfig(LoggingManager.java:148)
        at org.apache.jorphan.logging.LoggingManager.initializeLogging(LoggingManager.java:114)
        at org.apache.jmeter.util.JMeterUtils.getProperties(JMeterUtils.java:133)
        at org.apache.jmeter.JMeter.initializeProperties(JMeter.java:327)
        at org.apache.jmeter.JMeter.start(JMeter.java:242)
        at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
        at sun.reflect.NativeMethodAccessorImpl.invoke(Unknown Source)
        at sun.reflect.DelegatingMethodAccessorImpl.invoke(Unknown Source)
        at java.lang.reflect.Method.invoke(Unknown Source)
        at org.apache.jmeter.NewDriver.main(NewDriver.java:161)
```

My head was already hurting so badly that I didn't want to debug the problem further. JMeter logging code could have some clean-up work.

## Troubleshooting

If you get an NullPointerException about NOT_UNDEFINED property when creating your element in GUI, the BeanInfo class of your component has a wrong filename.

If widget labels are not read from resouces file, your the properties file of your component has a wrong filename

Email me