Unique Essbase Customizations Using Java – Custom Logging

Introduction

Many companies have in depth working knowledge of Hyperion Essbase and are looking to enhance their enterprise reporting capabilities to the next level. Companies typically have specific processes and calculations that set them apart in their industry. However, they are often limited to basic reporting capabilities provided by the standard functions in Essbase. Additionally, complex operations can quickly become arduous using Calculation Scripts and Business Rules. This post will demonstrate the how to easily build Custom Java Routines to extend Essbase and dramatically reduce development time.

Complete details will be provided on how to implement a simple customized logging function for use in Calculation Scripts and Business Rules. Essbase's streamlined, parallel nature makes it difficult for application developers to trace line by line. By using Java to implement a custom logging routine, one may use personalized log entries within their Essbase scripts. Consequently, developers can add tracing to their scripts and quickly determine how Essbase is approaching each calculation. Accordingly, application developers are able to see exactly how the script is being executed – providing quick debugging and faster development time. One powerful feature is to help determine block creation within FIX statements.

The first step to integrating a custom Java routine into Essbase is to write some simple Java code. It is very easy – the code does not have to include any special APIs for Essbase. During development, a few issues were encountered where Essbase was a bit picky about how the code is written.

Here are a few tips to help in getting started. These tips were gathered while doing real development, and it is best to follow at first, though you may revisit the items and find out what will work for you.

- Do not include the code in a package such as “com.company.product_name” – remove the “package” declarative at the top of the code
- Do not use the keyword “this” to refer to variables
- Do not overload methods
- Set all methods and variables to static

With these provisions in mind, the following code can be written to implement a custom logging routine.

CustomLoggerV2.java

```
import java.io.FileWriter;
import java.io.Writer;
import java.io.PrintWriter;
import java.io.StringWriter;
import java.util.Calendar;
import java.util.ArrayList;

public class CustomLoggerV2
{

    private static String logFile;

    public static int logFilterLevel;

    public static void setLogFilterLevel(int logFilterLevel2)
    {
        logFilterLevel = logFilterLevel2;
    }

    public static void setLogFilename(String logFilename)
    {
        logFilterLevel = 0;
        logFile = logFilename ;
    }
```

```
     public static synchronized void customLog (int logLevel,
String message)
    {
        log(logLevel, message);
    }

    private static synchronized void log (int logLevel, String
message)
    {



        // do not log
        if (logLevel < logFilterLevel)
            {
                return ;
            }

        try {

            Calendar c = Calendar.getInstance();

            FileWriter fw = new FileWriter(logFile, true);

            fw.write(c.getTime()   ": "   message   "\n");
            fw.close();

        } catch (Exception e)
            {
                 System.out.println("Error, cannot open , "
logFile);
                e.printStackTrace();
            }
        }


}
```

The code implements three public methods:

- setLogFilterLevel(int logFilterLevel) – sets the minimum message level to log (think about ERROR=100, WARN=90, INFO=70, DEBUG=0) – so you can easily change the

verbosity of the output.
- setLogFilename(String filename) – The full path to the log file you wish to use
- customLog(logLevel, String message) – The log message, with its indicated priority

The next step is to package up the code above. It is important to use the same version of Java which is running your Essbase instance. To find the version, look for the JRE being used within the environment, for instance, Hyperion\common\JRE\Sun\1.5.0\bin. To obtain the specific revision, open a cmd prompt, cd to the bin directory, and run "java –version".

```
E:\Hyperion\common\JRE\Sun\1.5.0\bin>java -versi

java version "1.5.0_11"

Java(TM) 2 Runtime Environment, Standard Edition (build 1.5.0_11-b03)

Java HotSpot(TM) Client VM (build 1.5.0_11-b03, mixed mode
```

To compile the code a JDK is required, which will contain the javac command. Hyperion only packages the JRE, meaning you will have to download the correct JDK in order to compile the code. You can find older versions of Java JDK from Oracle(Sun)'s web site. Once you have obtained the correct version of the JDK, compile and package up the code:

```
javac CustomLoggerV2.java

jar -cf CustomLoggerV2.jar CustomLoggerV2.class
```

Next, copy the CustomLoggerV2.jar file into the Essbase file structure:

Copy CustomLoggerV2.jar into the E:\Hyperion\products\Essbase\EssbaseServer\java\udf folder. If the udf folder does not already exist, create it.

Now it is time to start including the Java class within Essbase. Essbase runs within its own JVM and therefore has its own Java security. In the example above, we are writing to a local log file, which will violate the default security policy setup in the udf.policy file. The file is usually found in Hyperion\products\Essbase\EssbaseServer\java . The simplest way to get around the security concerns for development purposes is to remove the comment from the last line in the file, which effectively includes the directive “permission java.security.AllPermission”

```
…

permission java.util.PropertyPermission “java.vm.version”, “read”;

permission java.util.PropertyPermission “java.vm.vendor”, “read”;

permission java.util.PropertyPermission “java.vm.name”, “read”;

// Uncomment the following line if you want to remove all restrictions

permission java.security.AllPermission;

};
```

Now that the Essbase security and jar file have been put in, a restart of the Essbase process is required to register the changes. Please restart Essbase now.

The final step is to run some maxl statements to register the public java methods with Essbase.

CustomLoggerV2.mxl

```
create or replace function '@JCustomLoggerV2_setLogFilename'

as 'CustomLoggerV2.setLogFilename(String)'
```

```
spec '@JCustomLoggerV2.setLogFilename(absolute file name)'

comment 'Nicholas King'

with property runtime;

create or replace function '@JCustomLoggerV2_customLog'

as 'CustomLoggerV2.customLog(int, String)'

spec '@JCustomLoggerV2.customLog(log level, log message)'

comment 'Nicholas King'

with property runtime;


create or replace function '@JCustomLoggerV2_setLogFilterLevel'

as 'CustomLoggerV2.setLogFilterLevel(int)'

spec '@JCustomLoggerV2.setLogFilterLevel(filter level)'

comment 'Nicholas King'

with property runtime;
```

One final thing… In order to run a custom java function, the value of the result has to be stored in an Essbase member. This is true even if there is not any use for the return value, such as this case where there is no value returned from the Java methods. To get around this, create a new Essbase member called “No Measure” somewhere within your Essbase outline. This will act as a dummy member intended only to direct the return value, if any, of the Java methods. An example is shown below.

Sample Calc Script or Business Rule to Invoke the Logger

```
//ESS_LOCALE English_UnitedStates.Latin1@Binary
```

```
/* SETUP The Logger */

/* Fix on something so it runs only once */

FIX (Actual, Texas, "100-10")

"No Measure" = @JCustomLoggerV2_setLogFilename("E:\CustomEssbaseLog.log");

"No Measure" = @JCustomLoggerV2_setLogFilterLevel(50);

ENDFIX;

/* In your script, do some actual logging */

FIX (Actual, Texas, "100-10")

/* Won’t be displayed */

"No Measure" = @JCustomLoggerV2_customLog(0, "This is a debug message");

/* Will be displayed */

"No Measure" = @JCustomLoggerV2_customLog(50, "This is a normal message");

"No Measure" = @JCustomLoggerV2_customLog(100, "This is an important message");

ENDFIX;
```

The result of running the script is the log entries will be added to the log file E:\CustomEssbaseLog.log,

```
Mon Feb 21 01:30:25 EST 2011: This is a normal message

Mon Feb 21 01:30:25 EST 2011: This is an important message
```

Troubleshooting Tips

A very common error you may receive is,

Error: 1200324 Error compiling formula for [No Measure] (line 8): operator expected after [@JCustomLogger_customLog]

This error is a generic error that indicates something in your custom function is not registered properly. Unfortunately, there is not a lot of detailed log information at this point to help discover the problem. If you receive this message a few things might help:

- Retrace your steps – carefully review all instructions above
- Check that the correct version of Java was used to compile the class file and package the jar
- Check the jar is in the correct “udf” folder in Essbase
- Check the syntax of the MAXL to register the functions is correct
- Simplify your script as much as possible to reduce the possibility of syntax errors

Conclusion

This example shows how to create a custom Java based logger integrated into Essbase. The possibilities are endless – anything that can be done in Java can be added to Essbase. You can create development aids, or even read/modify the values within the cube. For instance, this model has successfully been used to perform complex financial calculations within Hyperion Planning Forms using Business Rules. It could also be used for integrating Web Services with your cube by reading or writing cube data and interacting with an enterprise Web Service.