RI PC Platform Logging (Quick Start)

In an effort to provide a flexible logging solution, we are incorporating the *log4c* library into the RI Platform repository.

log4c is part of a larger family of log4x logging utilities that originated after the Apache's log4j Java library. It supports categories, allowing to enable /disable modular logging as well as up to seven logging levels, each providing different verbosity and semantics. It also comes with a powerful library of appenders, which allow to customize the logging output to use different output facilities such as terminals, files or network protocols.

log4c is available under the LGPL 2.1 license. More information about the library can be found at this location.

RI Platform

You can configure the platform logging system by making changes to the *log4crc* file located in $PLATFORMROOT. The *log4rc* file is organized into three basic sections: layouts, appenders, and categories.

log4c Layouts

In *log4c*, a layout describes the formatting applied to each log message before it is output to its destination by an appender. The *log4c* base library provides two standard layout types. The ***basic*** layout contains the only the log priority, logging category, and the message:

```
INFO     RI.Display- create_display -- called
INFO     RI.Display- read_config_values -- called
INFO     RI.Display- create_test_pipeline -- called
INFO     RI.ATE- ate_FStrmThread waiting for a connection...
INFO     RI.ATE- ate_TelnetThread waiting for a connection...
INFO     RI.Pipeline.OOB- create_oob_pipeline() --
INFO     RI.UAL.ual- returning new GPNVS association ctx[0] 030C49B8:02859654
INFO     RI.CG- cg_upnp_controlpoint_subscribe(02859118, 0285AED4, 300);
```

The ***dated*** layout adds a timestamp to each line:

```
20090416 12:18:45.546 INFO     RI.Display- create_display -- called
20090416 12:18:45.546 INFO     RI.Display- read_config_values -- called
20090416 12:18:45.546 INFO     RI.Display- create_test_pipeline -- called
20090416 12:18:45.609 INFO     RI.ATE- ate_FStrmThread waiting for a connection...
20090416 12:18:45.609 INFO     RI.ATE- ate_TelnetThread waiting for a connection...
20090416 12:18:45.625 INFO     RI.Pipeline.OOB- create_oob_pipeline() --
20090416 12:18:45.625 INFO     RI.UAL.ual- returning new GPNVS association ctx[0] 030C49B8:02859654
20090416 12:18:45.625 INFO     RI.CG- cg_upnp_controlpoint_subscribe(02859118, 0285AED4, 300);
```

As you will see in the deafult *log4crc* file, the RI Platform actually use a custom version of each of these layouts, ***basic_nocr*** and ***dated_nocr***. The *nocr* stands for "no carriage return". The standard layouts defined in log4c automatically append a newline character to each log message. The custom layouts were necessary due to the fact that the native code in ODL stack already has newline characters explicitly specified in all of its logging messages. It was easier to define new layouts than to go and modify all of the existing ODL source code.

log4c Appenders

log4c appenders allow you customize where log messages are written. The *log4c* library comes with a standard appender called ***stream***. The stream appender allows log messages to written to any standard file descriptor -- this includes stdout and stderr. In the *log4crc* file, the appender type is listed as "stream" while the appender name describes the target (either stdout, stderr, or a filename). We have created some custom versions of the stream appender to support the needs of the RI Platform.

The ***stream_env*** appender allows you to use environment variables in the appender name. The system will parse the environment variable to construct the final path to the desired file:

```
<appender name="stderr" type="stream_env" layout="dated"/>
<appender name="$(PLATFORMROOT)/RILog.txt" type="stream_env" layout="dated"/>
```

The ***stream_env_plus_stdout*** appender will implicitly log all messages to stdout in addition to the given file name:

```
<appender name="$(PLATFORMROOT)/RILog.txt" type="stream_env_plus_stdout" layout="dated"/>
```

The ***stream_env_append*** apender will append log messages to the given log file instead of overwriting it:

```
<appender name="$(PLATFORMROOT)/RILog.txt" type="stream_env_append" layout="dated"/>
```

Finally, the ***stream_env_append_plus_stdout*** appender adds implicit stdout logging

```
<appender name="$(PLATFORMROOT)/RILog.txt" type="stream_env_append_plus_stdout" layout="dated"/>
```

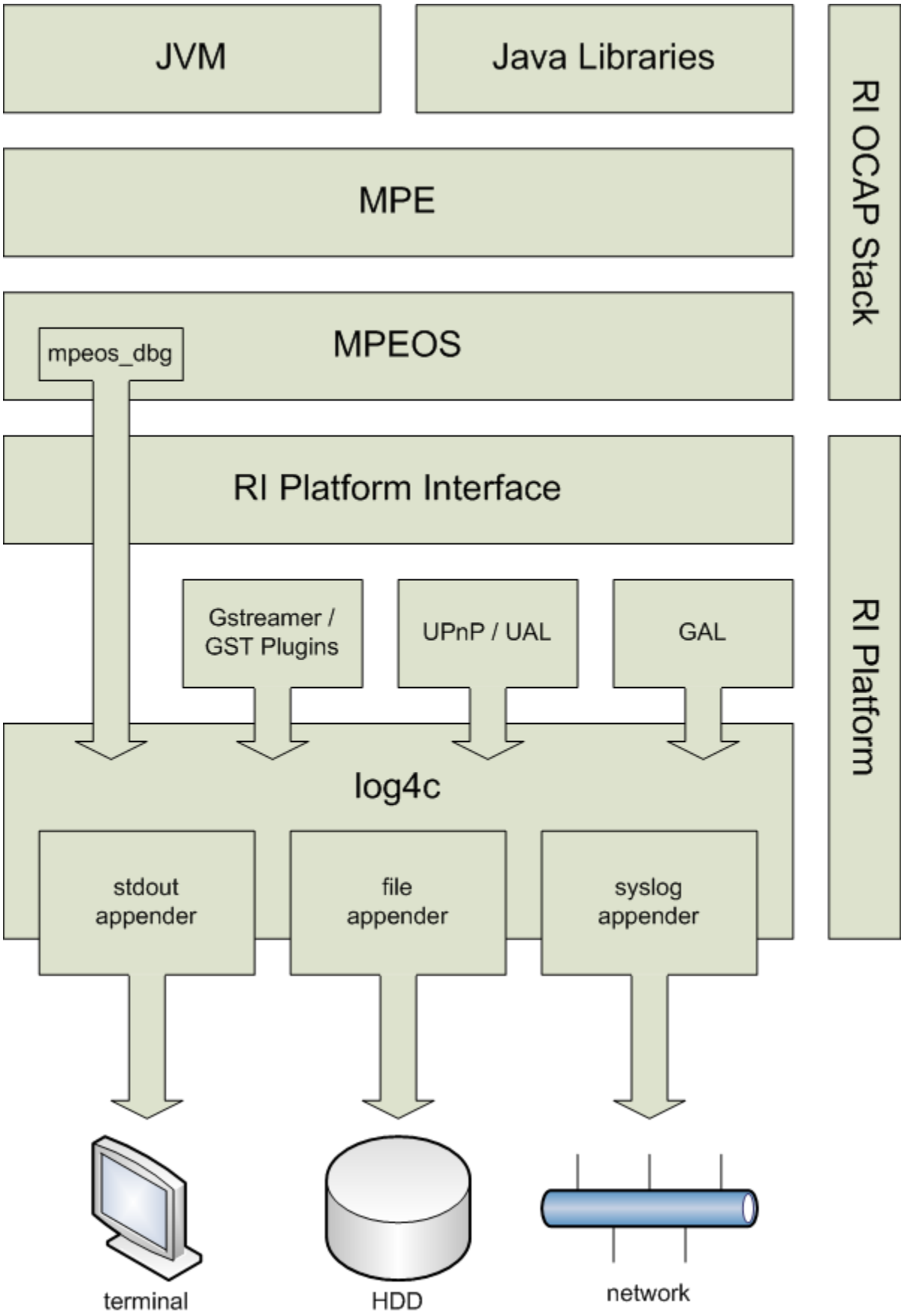
Categories

In *log4c*, categories allow you to break up log messages into logical groupings (usually based on functional areas). Initially, a number of categories will be defined in the RI Platform:

1. One category per RI Platform module implementation - such as *RI_TUNER*, *RI_SECTIONFILTER*, etc.
2. One "pass-thru" category for all messages coming through the mpeos_dbg function. At the present time, this will get us up and running quicker. If a more robust solution is needed, we can consider more separation between the platform and the stack as outlined in the **Alternative System Logging Solution** at the end of this page.
3. UAL - for all UPnP adaptation layer messages.
4. GLIB - for all of the non-GStreamer but Gnome library initiated debugs (which would include, for example, all of the utility function logging - GArrays, GLists, etc.)
5. GStreamer messages. A mapping to the GStreamer logging function will be created that is going to translate all of the internal GStreamer function calls into *log4c* equivalents. In particular, a custom GstLogFunction will be supplied to GStreamer to route all of its messages through *log4c*. The full GStreamer logging API can be found here. Ideally, this will encapsulate:

- GST_* prefix - GStreamer Core debug category. Includes items such as GST_BUS, GST_PIPELINE, GST_REFCOUNTING, GST_PADS and all other core GStreamer categories. A full list of them can be found here - defined inside the function *_gst_debug_init*.
- GStreamer Element categories - both existing (such as *udpsrc*, *rtpmp2tdepay*) and created by CableLabs (*sectionfilter*, *sectionsink*).

It is expected that each of them can be split into more categories as our code base grows and more features are added to the RI Platform. Conceptually, the logging will look as pictured below, with a "pass-thru" function exposed by the RI Interface to allow for logging of all of the stack messages. For local RI Stack/RI Platform logging, *stream* appenders should be sufficient.


JVM
Java Libraries
RI OCAP Stack
MPE
MPEOS
mpeos_dbg
RI Platform Interface
RI Platform
Gstreamer / GST Plugins
UPnP / UAL
GAL
log4c
stdout appender
file appender
syslog appender
terminal
HDD
network

RI Stack

Configuring Log4j

The RI Stack Java code uses log4j as its logging framework. The default log4j.properties is copied by the build process from $OCAPROOT/java/src/base /log4j.properties to $OCAPROOT/bin/$OCAPTC/env. The env folder is in the classpath, so changes to log4j.properties in this folder will result in a change to your log4j configuration. log4j.properties -used- to be included in $OCAPROOT/bin/$OCAPTC/env/sys/ocap-rez.jar, which makes it difficult to edit. Make sure the $OCAPROOT 'clean' ant target has been ran prior to modifying log4j.properties - otherwise the outdated ocap-rez.jar containing log4j.properties will be used instead of log4j.properties in the env folder, since ocap-rez.jar is ahead of the env folder in the classpath.
DEBUG logging is enabled by default in this configuration file, but two loggers are configured with a higher threshold: log4j.category.org.cybergarage.util. Debug=INFO log4j.category.org.cablelabs.impl.manager.application.AppEventQueue=FATAL

The cybergarage library used by the RI stack has been modified to output logging via the log4j framework. Cybergarage generates a large volume of messages at debug level, which is why the org.cybergarage.util.Debug logger is set to INFO in the default log4j.properties configuration.

If you need to see DEBUG severity logging from the cyberagarage library, either modify this file and rebuild, or modify the classpath defined in $OCAPROOT/bin/$OCAPTC/env/mpeenv.ini and copy a modified log4j.properties into the classpath(before ocap-rez.jar)

TODO: describe other logging support in mpeenv.ini