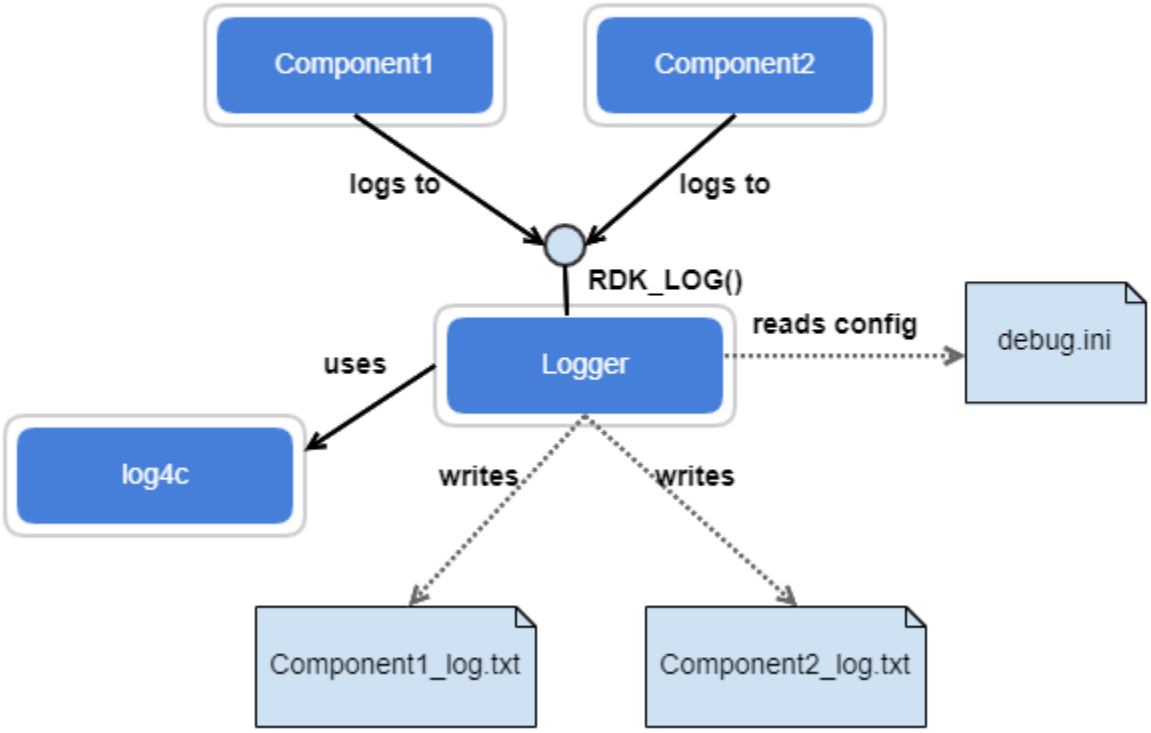
RDK Logger

RDK Logger is a common logging library which is based on MPEOS logging & it uses log4c for formatting and supports multiple log levels

RDK Logger Capabilities

- Abstracts logging client from underlying logging utility.
- Dynamically enables/disables logging level at run time.
- Provides logging format that complies with the existing OCAP format (e.g. <timestamp> [mod=*, level=*]).
- Controls log level independently for each component/module.
- Enables logging globally via single configuration value.
- Controls initial log level for each component/module from configuration file (debug.ini) at startup.
- Prints formatted data to stdout.
- Separates logs into separate files based on "**SEPARATE.LOGFILE.SUPPORT**" configuration variable

Architecture



Log4c – Introduction & Terminology:

- Framework which provides the capability to log messages in a variety of formats to console or local file
- Configuration - Log4c can be configured using a configuration file
    - log4crc keeps configurations in xml format
    - Configurations can be provided for modules, sub modules etc.
- Loggers - Named log message destinations which are known to applications.
- Appenders - Responsible for delivering LogEvents to their destination.
- Layouts & Filters determine how the message is formatted & filter the messages according to the configuration.

Logging Levels supported by RDK Logger

Code	Description
RDK_LOG_FATAL	Any error that is forcing a shutdown of the service or application to prevent data loss (or further data loss), reserve these only for the most heinous errors and situations where there is guaranteed to have been data corruption or loss.
RDK_LOG_ERROR	Any error which is fatal to the operation but not the service (cant open a file, missing data, etc)
RDK_LOG_WARN	Anything that can potentially cause application oddities, but for which the application automatically recovering.
RDK_LOG_NOTICE	Anything that largely superfluous for application-level logging.
RDK_LOG_INFO	Generally useful information to log (service start/stop, configuration assumptions, etc).
RDK_LOG_DEBUG	Information that is diagnostically helpful to people more than just developers.
RDK_LOG_TRACE1, RDK_LOG_TRACE2,...	Only when it would be "tracing" the code and trying to find one part of a function specifically.

How to initialize RDK Logger

- rdk_logger_init() loads entries from “debug.ini” and initializes log4c.
- Sets log level of the module corresponding to environment variables.
- Sets log level to default log levels, for the modules not having entries in configuration file. (Currently it is set as **LOG.RDK.DEFAULT = ERROR** in debug.ini)

How to add the logger functionality to the new module?

Include rdk_debug.h header file and make use of RDK_LOG for printing logs. Initialize RDK Logger by calling rdk_logger_init() in the respective module /component. Build new module/component by linking "librdkloggers.so" along with "liblog4c.so" and "libglib-2.0.so" shared object.

Example: -lrdkloggers -L ../opensource/lib -llog4c -lglib-2.0

RDK Logger Usage Format

RDK_LOG (rdk_LogLevel level, const char *module, const char *format,...)

- "level" is Log level of the log message (FATAL, ERROR etc). Apart from this there are special log levels like ALL, NONE, TRACE, !TRACE are supported.
- "module" is the Module to which this message belongs to (Use module name same as mentioned in debug.ini)
- "format" is a printf style string containing the log message.

All the RDK components logs are stored under /opt/log/ with a naming convention <RDK component>=""">_log.txt.
For example, /opt/log/pod_log.txt includes all events logged by POD Manager module.

Sample code for Logging

For example, add a debug messages for "INBSI" module

```
RDK_LOG (RDK_LOG_NOTICE, "LOG.RDK.INBSI","<%s: %s>: Sending PMT_ACQUIRED event\n", PSIMODULE, __FUNCTION__);
```

User needs to provide the module name "LOG.RDK.INBSI", which is the same as mentioned in debug.ini

```
$ cat debug.ini
EnableMPELog = TRUE
LOG.RDK.INBSI = ALL FATAL ERROR WARNING NOTICE INFO DEBUG
```

Sample logging output

```
131011-21:21:49.578394 [mod=INBSI, lvl=NOTICE] [tid=4141] <SITP_PSI: NotifyTableChanged>: Sending PMT_ACQUIRED event
```

In this way, user make use of the RDK logger in the respective modules and control the logging levels through configuration file. Here, No need to build RDK logger again for the addition of new components/module.

How to use logging in G-Streamer

A callback function gst_debug_add_log_function() is registered to receive GStreamer logs. Logs are converted to RDK logs in callback function. RMF element which controls a gst-element shall register element name and corresponding log module using

```
void RMF_registerGstElementDbgModule(char *gst_module, char *rmf_module)
```

Callback function uses this information to get module names corresponding to gstreamer logs.

RDK Logging Configuration

Default level for RDK logging is ERROR and logging settings are configured in **/etc/debug.ini.** RDK components reads the configuration details from config file at the beginning.

```
LOG.RDK.<component1> = FATAL ERROR WARNING NOTICE INFO
LOG.RDK.<component2> = FATAL ERROR WARNING NOTICE INFO DEBUG
```

For Example:

```
LOG.RDK.UI = NONE
LOG.RDK.QAMSRC = ALL FATAL ERROR
LOG.RDK.VODSRC = ALL FATAL ERROR WARNING NOTICE INFO
LOG.RDK.MS = ALL FATAL ERROR WARNING INFO
LOG.RDK.GSTQAM = ALL FATAL ERROR WARNING INFO
LOG.RDK.IPPV = ALL FATAL ERROR WARNING NOTICE INFO
LOG.RDK.RMFBASE = FATAL ERROR WARNING INFO
LOG.RDK.TRM = ALL FATAL ERROR WARNING NOTICE INFO
```

Which path logs are stored

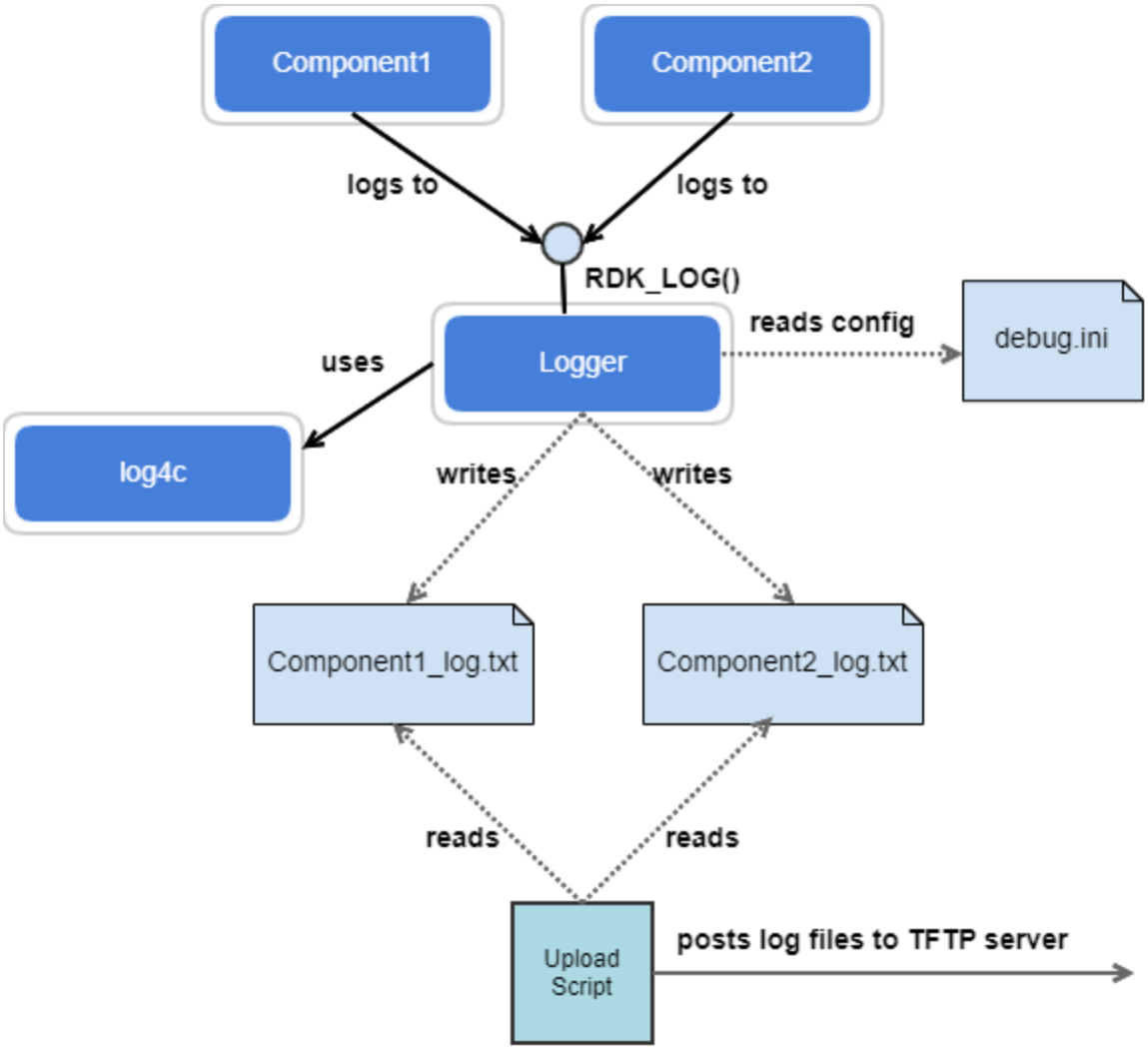
By default, stdout is redirected to **/opt/logs/ocapri_log.txt** for OCAP RI related logs.

RDK Log upload Process

Logs are uploaded to servers in following scenarios

- When the box is rebooted
- When an on-demand log request is made
- When a regular nightly job is set up for the log uploads

the script uploadSTBLogs.sh is responsible for uploading the logs to the server. Logs are uploaded as tar files with sufficient information like Mac ID, date , timestamp. Log can be uploaded to server using scp or tftp protocol as per server requirement.



API Documentation

To know more about SoC/Application level APIs details use in RDK, refer the link RDK LOGGER API Documentation