Messaging for a Dynamic World

Mobeon Internal

Approved: Per Berggren	No: 2/FS-CRH 109 581-1 Uen

Copyright Mobeon AB All rights reserved	Author: Marcus Haglund Title: FS-Log Manager	Version: PA1 Date: 2007-08-27	1/11

FS –Log Manager

Content

1 History

Version	Date	Adjustments
PA1	2007-08-27	First version of the document. Original document

		was found in MAS. (EMAHAGL)

2 Introduction

This document specifies the Log component in MAS. It provides an interface that can be used from other components in the MAS in order to generate log entries. The log entries contain information that, depending on severity, may end up in a log file or be forwarded to a log server. Log entries must be applicable to ref [1].

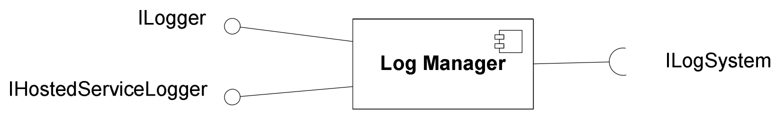

Figure 1 Log Manager component

2.1 Glossary

Category	Grouping of source locations, from where a Log Entry can originate. In most cases a Category corresponds to a class package or a set of class packages. The grouping and hierarchy of Categories are configurable.
Exception	A Java or C++ Exception, carrying error information such as error type, error message and call stack information.
Log	A collection of Log Entries. In its simples form, the Log is a text file where information from the Log Entries is written.
Log Entry	An object containing information that shall be logged. In many cases, the object is simply a textual string.
Log Level	A fixed set of levels, used to classify the severity of a Log Entry. The following Log Levels exist: - ALL - DEBUG - INFO - WARN - ERROR - FATAL - OFF The levels may be seen as thresholds use to determine if a Log Entry should be forwarded to the Log or not.

Hosted Service	A service following a protocol and resides at a host available on a specific port.

3 Function Requirements (Commercial)

3.1 Log Levels

The Log component allows the application developer to generate Log Entries with different severity. Some entries are only of interest during development, others are vital for locating errors during live operation of the system.

It is possible to filter which entries are being forwarded to the Log by setting a configurable rule, stating the lowest required severity level an entry must have in order to be forwarded to the Log.

The available log levels are DEBUG, INFO, WARN, ERROR and FATAL.

3.1.1 Log Level severity order

The log levels are ordered in a severity order, where DEBUG is the least severe level and FATAL is the most severe level.

During runtime, it is possible to determine what severity level that shall be the threshold for a log item to be forwarded to the log file. E.g. if the filter is set to WARN, all log items with a severity of at least WARN will be forwarded, but not DEGUG, since those items have a severity less that WARN.

3.2 Different Log Levels per sub-component

The Log component allows for different Log Levels to be specified for different sub-component. A sub-component is defined by the developer and can be as fine-grained as a specific class.

In general each component, identified as a class package, has a Log Level definition.

3.3 Change Log Levels during run-time

It is possible to change the Log Level rules dynamically during runtime without restarting the MAS.

3.4 Trace User Sessions

It is possible to follow a specific user session by specifying the user session identity in a configuration file. Any number of users can be specified at the same time.

Session data, used to identify a user session, e.g. A-number, B-number etc. is registered with the Log component by the system when a new session is created.

A configuration is used to inform the Log component that a User Session should be traced, and what data should be used to find Log Events related to a specific session.

If trace of a specific user is enabled, only Log Events originating from that user sessions, and/or events from the Log Levels Error and Fatal are forwarded to the Log.

It is possible to enable/disable session tracing dynamically without restarting the MAS.

3.5 Log Handling

It is possible to specify how a Log shall be managed, e.g. if the Log Entries should be written to a file, if the file is of "circular" nature (Once it reaches a specific size, it starts overwriting the oldest parts of it self), if a new log file should b created each day, if the Log Entries should be send to a centralized log server etc.

In the default setup, the Log is aged in 10 generations. A new generation is started when the Log reaches the size limit, 10 MB.

3.6 Hosted service availability

Repeated logging of hosted service circumstances should be avoided. E.g. if a hosted service is considered not available by a client, the client notifies the log system about these circumstances. All consecutive notifications in the same context for the same hosted service should be filtered and not logged.

Clients shall always notify the log system that a hosted service is available. This is the normal case. The log system shall filter and not log this information if no preceding other circumstances is logged for that hosted service. But if the log system is notified that a hosted service is not responding or not available and becomes available again, this should be logged.

Clients shall always notify the log system that a hosted service is not responding while trying to connect to or use it. Typically clients keep trying the hosted service until it is considered not available. These circumstances shall only be logged if it not already known that a hosted service is not responding or is not available.

Clients shall always notify the log system that a hosted service is not available. Typically no further tries to connect to or use it will be executed. These circumstances shall only be logged if it not already known that a hosted service is not available.

The consecutive logging filter shall also consider calls from two different client instances as consecutive calls.

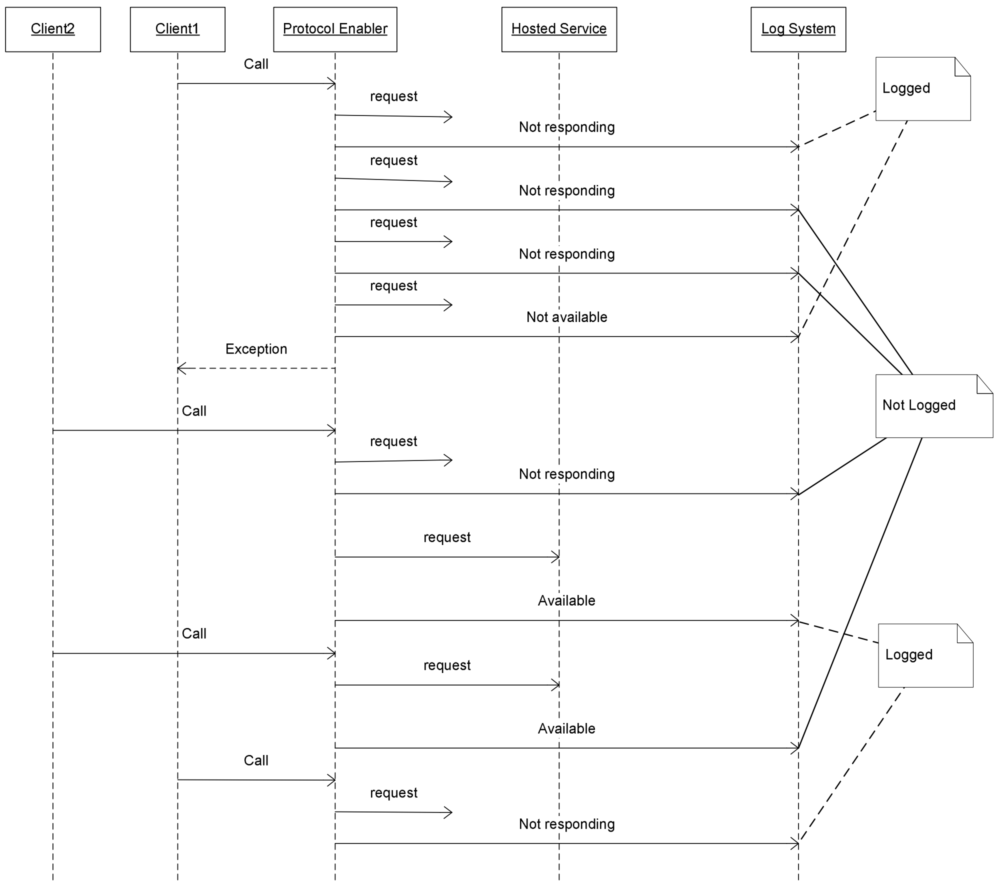

Figure 1 Example of logging hosted service availability

Example of logging hosted service availability:

1. Client1 calls protocol enabler.
2. Protocol enabler requests hosted service with no response. Protocol enabler notifies log system that hosted service is not responding. Notification **is logged** as a warning.
3. Protocol enabler tries again requesting hosted service with no response. Protocol enabler notifies log system that hosted service is not responding. Notification **is <u>not</u> logged**.
4. After some number of tries the protocol enabler considers the hosted service not available. Protocol enabler notifies log system that hosted service is not available. Notification **is logged** as an error. Protocol enabler signals to Client1 that hosted service is not available.
5. Another client, Client2, calls protocol enabler.
6. Protocol enabler requests hosted service with no response. Protocol enabler notifies log system that hosted service is not responding. Notification **is <u>not</u> logged**, because it already "known" that hosted service not responses.
7. Protocol enabler tries again requesting hosted service with response. Protocol enabler notifies log system that hosted service is available. Notification **is logged** as information.
8. Client2, calls protocol enabler.
9. Protocol enabler tries requesting hosted service with response. Protocol enabler notifies log system that hosted service is available. Notification **is <u>not</u> logged** as information, because it already "known" that hosted service is available again.
10. Client1 calls protocol enabler.
11. Protocol enabler requests hosted service with no response. Protocol enabler notifies log system that hosted service is not responding. Notification **is logged** as a warning, because latest "known" state of the hosted service was that it was available.

4 Function Specification (Design Related)

4.1 Java and C++ bindings

The Log component provides language bindings for both Java and C++, exposing the same interface in both environments.

4.2 Source code reference in the Log Event

The Log Event contains information related to the Source Code, i.e. the location in the source files where the Log Event was issued.

4.3 Exception information

It is possible to add Exception information with the Log Event. The information carried by the Exception is then forwarded to the Log.

4.4 Possibility to check if logging is enabled

It is possible to check if debug level logging is enabled before a call to the Log interface is performed. This could be done in order to achieve better runtime performance since it allows not incurring the cost of parameter construction if debugging is disabled

4.5 Log object as a static class member

The interface to the Log component is available as a static class member in each class wanting to send Log Events to the Log.

4.6 Log Presentation Format

It is possible to specify format in which a Log Entry is presented/stored using a configuration file. The format is limited only by the data available in the Log Event.

In the default configuration the following log format is used:

<timestamp><source><loglevel>[PID:<pid>][SID:<sid>]<trace>

Timestamp	Time of the log entry conformant with ISO 8601:2000 with the format as yyyy-mm-dd hh:mm:ss,msec timezone. Time zone is specified as +-hhmm offset from UTC.
Source	Reference to the component responsible for the Log Entry.
Loglevel	One of DEBUG, INFO, WARN, ERROR, FATAL
Pid	The process id of the process originating the log.
Sid	End-user session Id.
Trace	The message to log.

Example:

```
2005-06-07 13:24:53:999 com.mobeon.masp.mailbox.IMAPImpl DEBUG
[PID:6371] [SID:123] – This is a debug message.
```

4.7 Log files may not be placed on certain locations

The log file must be placed in a location where it no harm can be made to the system, regardless of the size of the log file. Depending on operation system, filling up some file system may cause the whole node to cease working. E.g. on Solaris, /tmp are such a file system where a log file may not be placed.

4.8 Exported interfaces

The Log component export one interface, ILogger, used to access the log functionality.

4.8.1 The Logging interface (ILogger)

4.8.1.1 Functions

4.8.1.1.1 debug(Msg)

This method shall be used for generating log statements useful for debugging of the application. In general, these log items will be filtered out in commercial operation of the system.

4.8.1.1.2 info(Msg)

This method shall be used for generation of log items that may be of interest during commercial operation, but not indicating an error condition, e.g. logging of the number of currently allocated channels, duration of a call etc.

4.8.1.1.3 warn(Msg)

This method shall be used for generation of log items indicating a condition that is potentially harmful for the application, e.g. some resource within the system is reaching an exhaustion limit, the duration of a call has exceeded some expected time limit etc.

4.8.1.1.4 error(Msg)

This method shall be used for generation of log items indicating an error condition. The error is not fatal to the whole application, but may result in e.g. the termination of an end-user session or similar.

4.8.1.1.5 fatal(Msg)

This method shall be used for generation of log items indicating a fatal error. The application will terminate as a result of the occurred error.

4.8.1.1.6 registerSessionInfo(name, value)

This method shall be used t o associate a named value, e.g. an A-number, to the current session. The name and value is used by the Log component for filtering log items based on session identity. I.e. if an A-number has been registered with the Log component, the A-number can be used to filter out log items related to that session.

4.8.1.1.7 isDebugEnabled()

This method shall be used to verify if the debug log level is enabled. This method may be used to short-circuit the log handling functionality if the log call to be made is of the debug kind and the debug log level is not enabled. E.g.

```
If (logger.isDebugEnabled())
    logger.debug(“some message”);
```

4.8.2 IHostedServiceLogger

This interface provides methods specialized to log hosted service circumstances.

4.8.2.1 Methods

- *void notResponding(String protocol, String host, int port, String optionalMessage)*
  Notifies the log system that a hosted service is considered not responding by a client. Not responding notifications should be handled as a warning message. An optional message can be provided.
- *void notAvailable(String protocol, String host, int port, String optionalMessage)*
  Notifies the log system that a hosted service is considered not available by a client. Not available notifications should be handled as an error message. An optional message can be provided.
- *void available(String protocol, String host, int port)*
  Notifies the log system that a hosted service is considered available by a client. Available notifications should be handled as an info message.

5 External Operation Conditions

5.1 Configuration

The following should be configurable in Log Manager:

- Log level threshold (overall).
- Log level threshold per component.
- Log presentation format.
- Log output handling.
- Traced User Session A-number and B-number.

6 Capabilities

This paragraph is intentionally left blank.

7 References

[1] UCD Examine MoIP Log Files
9/155 53-1/HDB 10102 Uen

8 Terminology

UTC	Universal Time Coordinated.