Log & Trace Guidelines

Document purpose

Proper logging/tracing is an essential part for efficient fixing of tickets. The purpose of this document is to give a guideline about logging in general and logging with DLT in particular. This page should give some best practices, do & dont's and hint's for improving efficiency of your logging.

Generalized the whole topic can be summed up as: Log as little as possible but as much as needed. Thanks for the inputs from Damien Bernardeau, Marc Guillon, Stephan Knauss, Steen Larsen, Alexander Wenzel

General Topics

Think first

Before implementing logging in code one should take a second to think about a concept first. Often strategic places in the software can be used as a central place for logging. Such places are often interfaces to other SW components. Use the solution with the smallest impact. Avoid logging the "good cases" but log e.g. in your error handling sections – you will need error handling anyway. In case an error occurred more logs don't matter as long as your regular code produces little logs. Keep in mind that tracing comes with an pricetag – you are working in an embedded environment where CPU, memory and Bandwith are sparse.

Avoid high frequency outputs

Certain events occur very often in a system – some of them dozens of times per second. In such a case do not implement logging for each occurrence. One example is the screen frame rate. Instead of printing a log for each frame rate aggregate the information and print an average once every five seconds or – even better – report once a second if the frame rate is below a critical value.

Combine multiple messages

Please always consider that each Log message creates a certain overhead. In case of DLT as the way of logging each has a header of 20 bytes. Therefore please aggregate information. In this way all necessary information is always combined. Please always use a human readable format, use identifiers for the different values an be consistent with separators. This helps a great to work with the data, especially when log messages are processed by scripts. Such scripts often use reg ex expressions – make the job easier!

For example don't write log entries like this:

```
Total frames:  1000
Sync frames:   0
Reem frames:   0
Valid frames:  100
Urgent frames: 0
```

Better aggregate Information like this:

```
Frame info: total=1000, sync=0, reem=1000, valid=0, urgent=1
```

Do not use ASCII-art

Information should be "on your fingertips". Logging is a tool to ease crushing bugs, not to win a computer art contest. Don't use ASCII Art!

Do not create charts using ASCII

Charts can be a great help to visualize what is going on in the system. This type can be nicely done by a trace analysis or in case of usage of the DLTviewer, in a Plugin. It certainly should always be done in a post processing step. Doing this on the target is a waste of resources

Avoid tracing in loops

```
// bad implementation
for(int index=0; index<MAX; index++)
{
  LOG_MSG(("Loop: %d", index));
  /* ... */
}

// good implementation
for(int index=0; index<MAX; index++)
{
  /* ... */
}
LOG_MSG(("Loop count: %d", index));
```

Other do and avoids

Topic	Description
Avoid timestamps	Do not include a timestamp in your log messages. In case of DLT the logging system itself already provides a timestamp.
Avoid high logging at startup	Especially the system startup is always a high load situation. Please avoid to write a log of log outputs during the startup because the helps to avoid a overload situation and speeds up the startup.
Remove old log messages	If certain long log messages are not necessary anymore because the problem has been resolved please remove them from your code.
Do use constant separators	Use consistent field separators and delimiters between different variables and information in order to facilitate automatic log analysis.
Do use proper logs	The logs should contain the information that you require to identify a problem. If you find yourself producing a new binary to identify a problem every time somebody reports an error something may be wrong.
Avoid eye catchers	Please don't use custom highlighting for marking the messages which are important for you. The best way is to write a clear identifier in the front of your trace message. This can than (in the viewer) be easily used for filter sets or even for coloring. An example for messages informing about the frame rate could be: **Frame rate:** low: 12, high: 34, avr: 28
Do not log to the console	Do not use *printf()* or similar statements to trace to the console. Such mehaviour can make it problematik to work e.g. with the serial console or even might slow down the execution of the program. In a vehicle based testsystem the console messages are not recorded and will not help you.

The use of log levels

Loglevels wich are available in DLT

log level	description

DLT_LOG_FATAL	Fatal system errors, should be very rare
DLT_LOG_ERROR	Error with impact on correct functionality
DLT_LOG_WARN	Warning if correct behavior cannot not be ensured
DLT_LOG_INFO	Informational, providing high level understanding
DLT_LOG_DEBUG	Detailed debug information for programmers
DLT_LOG_VERBOSE	Verbose debug information for programmers

Please be aware that in car test systems normally run in the log level Info: This means message which are logged in info, warn, error and fatal will logged.

What to log at FATAL level

Fatal errors are the most serious error and should be very rare. They are, for example:

- Errors that cause the whole system to fail
- A corrupted boot environment which prevents system boot
- A critical hardware component is missing, failing or is preventing start-up.
- When your software/process/component exits due to a fatal error. Log the EXIT and the reason.
- Failure of a major critical component to start

What to log at ERROR level

This level is reserved for errors which impact the correct functionality of the system or its components. Errors related to connected customer devices such as phones should be logged at INFO level. Error level logs may be:

- A non-critical component is failing or cannot be found
- A system component is crashing
- An system essential file can't be read or written
- Detection of corrupted network messages, files, etc. when these impact correct
- Some major functionality could not be provided (e.g. the route in the navigation could not be calculated)
- When your software/process/component exits due to an error. Log the EXIT and reason

What to log at WARNING level

This level must be used for problems where a correct behavior cannot be ensured, i.e. problems that could affect the correct functionality of the system or its components. Warnings related to connected customer devices such as phones must be logged at INFO level. Examples for warning log messages could be:

- Vmost congestions
- DLT dropping logs
- No disk space for available for coredump
- Audio stream packet dropped
- If a process of calculation takes longer than the time allowed in specification e.g. Calculation of route in the navigation takes longer than allowed

What to log at INFO level

This level is reserved for key information and high-level events which are not errors or warnings of the system itself or of connected consumer devices.

- Start and non-error related stop of software components. Include version information in start log.
- Detection of key hardware components. Include key HW information in log.
- Customer device connected. Include key device and media info.
- Customer device detached or connection lost.
- Failure to connect to customer device. Include reason
- Corrupted disk, song, photo, etc. on customer device.
- Key system/HW information at start-up
- Information needed for reproducing and understanding user activity
- Information for reproducing the environment (Large volume data such as GPS traces should be logged at a reasonable rate. Especially with very frequent logs it should be taken care of that no redundancy occurs)
- Key information used for KPI (Key performance index) reporting

What to log at DEBUG level

This level should be used for debug information that can help developers debug the functionality of their software. For example:

- Information about entering and exiting major procedures
- Values of key variables, but not dumps of arrays and large number of variables
- Information about events received
- Network connection information
- Debug relevant information about hardware

What to log at VERBOSE level

This level is the most detailed level and should be used for in depth debug information that can help developers debug the functionality of their software. For example

- Detailed trace information
- Dumps of a large number of variables, dumps of arrays and structures
- Detailed information about events received, even events that happen very frequently
- Detailed network connection information
- Detailed hardware information
- Information about loops and iterations

Some information regarding the practical usage of DLT and code examples can be found here DLT usage guide and examples.

> Copied from MediaWiki
> Last Edit: 13:31, 13 January 2014 Birk.bremer