DLT cheatsheet

Authored by Alexander Wenzel

0.0.1, 2012/10/11: Initial version

Overview

DLT is quite easy to use. As an application developer you only need to follow some steps as described in this document.

Initialization:

- Link your application against the DLT library
- Include the DLT header file
- Register your application
- Define all contexts
- Register all contexts

Now you are ready to write your logs.

Termination:

- Unregister all contexts
- Unregister application

Link your application to DLT library
If you compile your application with cmake, you have to add the following lines to your CMakeLists.txt;

```
find_package(PkgConfig)

pkg_check_modules(DLT REQUIRED automotive-dlt)
```

and use the variables $\{DLT_INCLUDE_DIRS\} and $\{DLT_LIBRARIES\} for the DLT include and library paths.

Include the DLT Header

To use DLT you have to include the DLT header file in each file you want to use DLT.

```
#include <dlt.h>
```

Register your application

You have to register your application as early as possible during the initialisation of your application. You have to call the DLT_REGISTER_APP(). It is only allowed to call DLT_REGISTER_APP() **once** in your application. You have to provide a application id, which size is maximum four charcters long. In this example we use "MAPP". And you can provide also a description for your application, here it is "Test Application for Logging".

```
int main(int argc, const char* argv[])
    {
      DLT_REGISTER_APP("MAPP","Test Application for Logging");

    }
```

- If your application uses fork(), you may not call DLT_REGISTER_APP before fork(). And fork() should never be called after DLT_REGISTER_APP. This is because of state information and inter process communication channel to daemon would be copied to new process, but threads would be not. If you are not sure where you are calling DLT_REGISTER_APP() the first time, you can initialise the DLT user library by calling the initialisation routine directly.

```
dlt_user_init();
```

- DLT_REGISTER_APP is asynchronous. It may take some milliseconds to establish the IPC channel. Because of this, you might lose messages if you log immediately after registering. Typically this is not a problem, but may arise especially with simple examples.

Define all contexts

You can create as many contexts as you like. You can declare one or more contexts in a C or CPP file. Each context is only allowed to be declared once. You have to provide a unique variable name for your context.

```
#include <dlt.h>

    DLT_DECLARE_CONTEXT(myContext1);
    DLT_DECLARE_CONTEXT(myContext2);
    DLT_DECLARE_CONTEXT(myContext3);
```

If you want to use a context in another C or CPP file, you can import the context by calling

```
#include <dlt.h>

    DLT_IMPORT_CONTEXT(myContext1);
    DLT_IMPORT_CONTEXT(myContext2);
    DLT_IMPORT_CONTEXT(myContext3);
```

Register all contexts

After you have registered your application you must register your contexts, early during initialisation of your application. Do not call DLT_REGISTER_CONTEXT() before DLT_REGISTER_APP(). During registration of each context, you must provide a context id, which size is maximum four charcters long. In this example we use "TESX". And you can provide also a description for your context, here it is "Test Context X for Logging".

```
int main(int argc, const char* argv[])
    {
      DLT_REGISTER_APP("MAPP","Test Application for Logging");

      DLT_REGISTER_CONTEXT(myContext1,"TES1","Test Context 1 for Logging");
      DLT_REGISTER_CONTEXT(myContext2,"TES2","Test Context 2 for Logging");
      DLT_REGISTER_CONTEXT(myContext3,"TES3","Test Context 3 for Logging");

    }
```

Create your logs

Now you can start creating your logs. Each log command consist of the context, the log level and a variable number of logging parameters.

Log Level

The log level must be one of the following values:

Log level	Description
DLT_LOG_FATAL	fatal system error
DLT_LOG_ERROR	error with impact to correct functionality
DLT_LOG_WARN	warning, correct behaviour could not be ensured
DLT_LOG_INFO	informational (default)
DLT_LOG_DEBUG	debug
DLT_LOG_VERBOSE	highest grade of information

DLT_LOG_FATAL, DLT_LOG_ERROR and DLT_LOG_WARN should be used in your application, when something is going wrong. DLT_LOG_INFO should be used to send the most important information. DLT_LOG_DEBUG and DLT_LOG_VERBOSE should be only used for testing information.

Each context is set by default to DLT_LOG_INFO log level. All log message are send, which use this loglevel or a lower loglevel. If you also want to see DLT_LOG_DEBUG and DLT_LOG_VERBOSE log messages, you have to raise the log level with the DLT viewer.

Logging parameters

The following parameter types can be used. You can add one or more parameters to a single log message. The size of all logging parameters together should not exceed 2kBytes, including the DLT message header.

Type	Description
DLT_STRING(TEXT)	String
DLT_RAW(BUF,LENGTH)	Raw buffer
DLT_INT(VAR)	Integer variable, dependent on platform
DLT_INT16(VAR)	Integer 16 Bit variable
DLT_INT32(VAR)	Integer 32 Bit variable
DLT_INT64(VAR)	Integer 64 bit variable
DLT_UINT(VAR)	Unsigned integer variable

DLT_UINT16(VAR)	Unsigned 16 Bit integer variable
DLT_UINT32(VAR)	Unsigned 32 Bit integer variable
DLT_UINT64(VAR)	Unsigned 64 bit integer variable
DLT_BOOL(VAR)	Boolean variable
DLT_FLOAT32(VAR)	Float 32 Bit variable
DLT_FLOAT64(VAR)	Float 64 Bit variable

Logging command

Here are some examples for complete log messages. The contexts must be registered before.

```
DLT_LOG(myContext1,DLT_LOG_ERROR,DLT_INT(5),DLT_STRING("This is a error"));
    DLT_LOG(myContext2,DLT_LOG_INFO,DLT_INT(5),DLT_STRING("But this only information"));
    DLT_LOG(myContext3,DLT_LOG_DEBUG,DLT_INT(5),DLT_STRING("But this only information"));
```

Unregister contexts and applications

Before terminating your application you must unregister all registered contexts and unregister at last your application.

```
int main(int argc, const char\* argv\[\])
    {
      DLT_REGISTER_APP("MAPP","Test Application for Logging");

      DLT_REGISTER_CONTEXT(myContext1,"TES1","Test Context 1 for Logging");
      DLT_REGISTER_CONTEXT(myContext2,"TES2","Test Context 2 for Logging");
      DLT_REGISTER_CONTEXT(myContext3,"TES3","Test Context 3 for Logging");

      DLT_UNREGISTER_CONTEXT(myContext1);
      DLT_UNREGISTER_CONTEXT(myContext2);
      DLT_UNREGISTER_CONTEXT(myContext3);

      DLT_UNREGISTER_APP();

      return 0;
    }
```

Example

Finally here is a complete example for using DLT:

dlt_example.c.

```
#include <stdio.h>
    #include <dlt.h>

    DLT_DECLARE_CONTEXT(myContext1);
    DLT_DECLARE_CONTEXT(myContext2);
    DLT_DECLARE_CONTEXT(myContext3);

    int main()
    {
      /* register application */
      DLT_REGISTER_APP("MAPP","Test Application for Logging");

      /* register all contexts */
      DLT_REGISTER_CONTEXT(myContext1,"TES1","Test Context 1 for Logging");
      DLT_REGISTER_CONTEXT(myContext2,"TES2","Test Context 2 for Logging");
      DLT_REGISTER_CONTEXT(myContext3,"TES3","Test Context 3 for Logging");

      /* Write your logs */
      DLT_LOG(myContext1,DLT_LOG_ERROR,DLT_INT(5),DLT_STRING("This is a error"));
      DLT_LOG(myContext2,DLT_LOG_INFO,DLT_INT(5),DLT_STRING("But this only information"));
      DLT_LOG(myContext3,DLT_LOG_DEBUG,DLT_INT(5),DLT_STRING("But this only information"));

      /* Sleep some time to avoid a flaw in dlt-daemon that would eat your messages
         if you deregister while it still processes your registration */
      sleep(3);

      /* unregister your contexts */
      DLT_UNREGISTER_CONTEXT(myContext1);
      DLT_UNREGISTER_CONTEXT(myContext2);
      DLT_UNREGISTER_CONTEXT(myContext3);

      /* unregister your application */
      DLT_UNREGISTER_APP();

      return 0;

    }
```

CMakeLists.txt.

```
cmake_minimum_required(VERSION 2.6)

    find_package(PkgConfig)
    pkg_check_modules(DLT REQUIRED automotive-dlt)

    include_directories("${DLT_INCLUDE_DIRS}")

    project(DLTexample)
    add_executable(dlt_example dlt_example.c)

    target_link_libraries(dlt_example ${DLT_LIBRARIES})
```

Build steps.

```
mkdir build
    cd build
    cmake ..
    make
```