FAW HS7

IVI App Reference

LOG

Released on the: 2016/03/29

Version Number: 0.10

Neusoft Automotives ChangChun

**Version History**

| No. | Version | Section | Brief Description | Date | Responsible |
| --- | --- | --- | --- | --- | --- |
| 1 | 0.100 | All | Initial Create | 2016-03-29 | - |
|  |  |  |  |  | - |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

Content

[1 LOG User Spec 4](#_Toc447270594)

[1.1 Goal 4](#_Toc447270595)

[1.2 Specification 4](#_Toc447270596)

[1.3 ID Definitions 4](#_Toc447270597)

[1.3.1 ECU ID 5](#_Toc447270598)

[1.3.2 Application ID 5](#_Toc447270599)

[1.3.3 Context ID 6](#_Toc447270600)

[2 Code Reference 7](#_Toc447270601)

[2.1 Software Framework 7](#_Toc447270602)

[2.2 Working Flow 7](#_Toc447270603)

[2.2.1 Coding 8](#_Toc447270604)

[2.2.2 编译 10](#_Toc447270605)

# LOG User Spec

## Goal

为了便于开发人员在开发过程中或售后进行程序运行状况分析或错误分析，系统应该能够提供以下类型的日志：

* 错误日志
* 消息日志，记录程序进程间通讯信息
* 性能日志，记录系统的性能信息

## Specification

本节用于定义通用的LOG输出规范，所有新开发模块需要遵守。

* 带界面的应用程序，输出的AppID信息必须满4字节，且最后一位为固定数字。0表示前屏、1表示左屏RSE、2表示右屏RSE。例如：Video在左屏RSE上运行时，AppID为“VID1”。
* 进程开启时，需要输出固定字符串“Hello, World!”（注意：逗号后面有一个空格）。
* 进程退出时，需要输出固定字符串“Bye, World!” （注意：逗号后面有一个空格）。
* 在所有异常切不可复归分支（程序无法继续执行），必须输出Error级别的LOG。在文本中给出错误内容、出错原因等信息。
* 在所有异常但可以用默认值等方式保证程序继续运行（但不确保运行结果正确性）的分支，必须输出Warning级别的LOG。在文本中给出错误内容、出错原因等信息。
* 在各个函数的入口和出口输出VERBOSE级别的LOG。文本信息格式为“<function-name> IN/OUT”以及关键参数。
* 在一些重要的流程分支可以输出INFO级别的LOG。但需要确保不能滥用。
* 在一些调试需要的分支输出DEBUG级别的LOG。
* LOG输出的文本必须是可读的。即要求非本模块开发者也可以从LOG中获取足够信息。严禁为了突出自己的LOG而输出类似“=======”这样的无意义内容。
* LOG不允许快速重复出现导致刷屏影响其他LOG查看。如果本条目与前面的条目冲突，WARNING以上的LOG正常输出，其他LOG以本条为准。

## ID Definitions

### ECU ID

|  |  |  |
| --- | --- | --- |
| NO | ID | Comments |
| 1 | IVI | 娱乐系统ECU ID |
| 2 | CLST | 仪表盘的ECU ID |

### Application ID

|  |  |
| --- | --- |
| ID | Comments |
| AUDC | Audio Controller |
| AUDM | Audio Manager |
| BEEP | Beep Daemon |
| BTAA | Bluetooth Audio Application |
| BTPA | Bluetooth Phone Application |
| BTS | Bluetooth Service |
| CAMA | Camera Application |
| CAMS | Camera Service |
| CMNS | Common Service |
| EMDA | Engineer Mode Application |
| H5A | HTML5 Application |
| HMIC | HMI Controller |
| IMGA | Media Image Application |
| LAE | LibAudioEffect |
| LCYC | LifeCycle Library |
| MHS | MediaHub Service |
| MUSA | Media Music Application |
| OSMD | RPMSG-RPC Daemon |
| OSMS | RPMSG-RPC SDK |
| PKGM | Package Manager |
| PLC | Process Lifecycle Controller |
| POWM | Power Manager |
| RADA | Radio Application |
| RADS | Radio Service |
| RPCD | UART-RPC Daemon |
| RPCS | UART-RPC SDK |
| SETA | Setting Application |
| VIDA | Media Video Application |
| VRA | VR Application |
| VRS | VR Service |
| WIFI | WiFi Service(T.B.D. existing?) |

（本表格需要等待APP划分完成之后进行更新）

### Context ID

Context ID不做统一限定。由各Application自行设计。

# Code Reference

## Software Framework

Applications

AppSDK.so

Genivi DLT Component

HMI

CORE

LOG系统主要有三个组件实现，分别是：

* Applications：直接调用AppSDK.so动态库进行初始化和log输出。
* AppSDK.so：是对Genivi的DLT组件的封装，方便App的使用。
* Genivi DLT Component：是Genivi的标准组件，是LOG实现的真正组件。

## Working Flow

Register application

Define all contexts

Register all contexts

Output Logs

Unregister all contexts

Unregister application

**Initialisation**

**Termination**

**Running**

要使用DLT系统输出错误日志和消息日志的信息，则需要在用户的Application中加入一些工作流程。

在Application的初始化阶段：

* Register application：向DLT系统注册Application。注册时，需要提供一个最大4字节的Application ID。
* Define all contexts：一个Application可以定义多个Contexts。用户可以根据子功能切分等不同的依据，将日志进行分类。Context实质上是一个变量，因此定义时需要注意其作用域。
* Register all contexts：定义好的Contexts需要注册之后才能使用。与注册Application类似。注册Context时，同样需要提供一个最大4字节的Context ID。

在Application的退出阶段：

* Unregister all contexts：注销所有在初始化阶段注册过的Contexts。
* Unregister application：注销注册过的Application。

### Coding

appLogDemo示例程序中的main.c

|  |
| --- |
| /\*\* @mainpage helloWorld - None  \*  \* @author wangqi <qi.wang-neu@neusoft.com>  \* @version 1.0.0  \*\*/  //<使用log功能需要引用的头文件>  #include <appSDK.h>  //<定义自己使用的context><是可选项>  DLT\_DECLARE\_CONTEXT(MY\_CXT);  /\*\*  \* Main class of project helloWorld  \*  \* @param argc the number of arguments  \* @param argv the arguments from the commandline  \* @returns exit code of the application  \*/  int main(int argc, char \*\*argv) {  //<初始化dlt组件，“DEMO”为4字节的appID，1表示支持本地打印>  //<此过程初始化了默认context，BIZ默认是“def”，UI是“ui”>  dltInit("DEMO",1);  //<注册自己的context，“mcxt”是4字节的contextID><是可选项>  dltRegisterContext(MY\_CXT,"mcxt");  //<BIZ使用logXXX()>  //<UI使用logXXXUI()>  logError("I'm logError with arg1");  logError("I'm logError with arg1"," arg2");  logError("I'm logError with arg1"," arg2"," arg3");  logError("I'm logError with arg1"," arg2"," arg3"," arg4");  logError("I'm logError with arg1"," arg2"," arg3"," arg4"," arg5");  logError("I'm logError with arg1"," arg2"," arg3"," arg4"," arg5"," arg6");  logError("I'm logError with arg1"," arg2"," arg3"," arg4"," arg5"," arg6"," arg7");  logError("I'm logError with arg1"," arg2"," arg3"," arg4"," arg5"," arg6"," arg7"," arg8");  logError("I'm logError with arg1"," arg2"," arg3"," arg4"," arg5"," arg6"," arg7"," arg8"," arg9");  logError(MY\_CXT,"I'm logError with my own context");  logErrorUI("I'm logErrorUI");  logWarn("I'm logWarn with default context");  logWarn(MY\_CXT,"I'm logWarn with my own context");  logWarnUI("I'm logWarnUI");  logInfo("I'm logInfo with default context");  logInfo(MY\_CXT,"I'm logInfo with my own context");  logInfoUI("I'm logInfoUI");  logDebug("I'm logDebug with default context");  logDebug(MY\_CXT,"I'm logDebug with my own context");  logDebugUI("I'm logDebugUI");  logVerbose("I'm logVerbose with default context");  logVerbose(MY\_CXT,"I'm logVerbose with my own context");  logVerboseUI("I'm logVerboseUI");  int8\_t number1 = -8;  uint8\_t number2 = 8;  int32\_t number3 = -32;  uint32\_t number4 = 32;  int64\_t number5 = -64;  uint64\_t number6 = 64;  std::string astring = "a string";  bool abool = true;  logInfo(number1,number2,number3,number4,number5,number6);  logInfo(astring,abool);  dltUnregisterContext(MY\_CXT);  return 0;  } |
|  |

### 编译

工程使用了cmake进行编译，CMakeLists文件：

|  |
| --- |
| cmake\_minimum\_required (VERSION 2.8.1)  ######## Project settings ########  #<制定工程名称>  PROJECT(appLogDemo)  SET(LICENSE "TBD")  ######## Build and include settings ########  #<使用FindPkgConfig工具>  include( FindPkgConfig )  #<寻找相关动态库>  pkg\_check\_modules(DLT REQUIRED automotive-dlt>=2.2.0)  pkg\_check\_modules(APPSDK REQUIRED libappSDK)  #<由于打印消息>  message(STATUS "APPSDK\_LIBRARIES\_DIRS: ${APPSDK\_LIBRARIES\_DIRS}")  #<制定编译头文件>  include\_directories(  inc  ${APPSDK\_INCLUDE\_DIRS}  ${DLT\_INCLUDE\_DIRS}  )  #<制定编译源文件>  file(GLOB SOURCES  "src/\*"  )  #<编译可执行程序>  add\_executable(  appLogDemo  ${SOURCES}  )  #<连接制定动态库>  TARGET\_LINK\_LIBRARIES(  appLogDemo  ${DLT\_LIBRARIES}  ${APPSDK\_LIBRARIES}  )  #<安装可执行文件>  ######## Install targets ########  INSTALL(TARGETS appLogDemo  RUNTIME DESTINATION /home/wangqi/workspace/rootfs/NFSrootfs/usr/bin  ) |

编译步骤：

#cd bulid

#cmake ..

#make

#make install