Console Framework
Developer's Guide

June 11th 2014

Jerry Johns
System Software Engineering - Firmware
jjohns@apple.com

Contents

1 Introduction

1.1 What is this about?

The console framework sits at the heart of EFI and manages all console traffic that gets generated in the system. This framework was recently over-hauled and introduced a wealth of functionality that is beneficial to both developers and users alike. This document outlines the various new APIs that have been introduced and provides guidelines and best-practices for them.

1.2 Pre-requisites

The core concepts introduced in the re-arch have been outlined in a companion document, the *Console Framework User Guide*, namely *Section 2: Framework Design*. It is assumed that the reader has read that before commencing this document.

2 Module Registration

This section dives into the registration API and how it should be used correctly for the various producers in EFI.

2.1 Legacy vs. Registered Sources

By default, every macho/image in EFI registers a source against the console framework. This was put in place to faciliate the transition from the old world to the new. These producers are termed *legacy sources* (happens implicitly as part of the call to `EfiInitializeDriverLib`). These sources do not export any debug masks or information about the source. In addition, the default name is set to the name of the image itself.

Registered sources on the other hand explicitly call `EfiCoreLogRegisterModule` to register a source against the console framework. It can register optional debug masks as well as a name for reference.

2.2 API

`EfiCoreLogRegisterModule`	
Registers a module with the console framework	
`ModuleLoggingInfo *ModuleInfo`	Pointer to a structure that encapsulates information about the source (has to be static/global or heap memory that isn't freed after)
`ConsoleSourceRef *RefKey`	Pointer to an opaque reference that will be updated to hold a unique key for that source
`BOOLEAN SetAsDefaultModule`	Sets this source as the default for the current image (See 3.2 *API* below for usage examples)
`BOOLEAN IsLibrary`	Used when registering from a library

`struct ModuleLoggingInfo`	
`CHAR8 *ModuleName`	Name of the Module
`DebugLogMaskInfo *MaskList`	Pointer to a list of masks
`UINTN NumMasks`	Number of masks in the list above
`UINT32 GlobalDebugLevels[1]`	Global debug mask mapping (just one level)

`struct DebugLogMaskInfo`	
`DebugLogMask Mask`	32-bit mask value
`CHAR8 *ModuleName`	Module Name
`UINTN NumMasks`	Number of masks

2.3 Usage

The `ModuleName` field should point to an ASCII name (in lower-case) that represents a chipset or functionality. `GlobalDebugLevels[0]` should be set to a valid debug-mask as befitting the source.

The `RefKey` passed in is an opaque reference that gets updated by the console framework. This key is to be later used with some of the logging functions (See 3.2 *API*).

2.4 Example

```
#define GG_DBGLVL_TYPE          (1 << 0)
#define GG_DBGLVL_PKTDATA       (1 << 1)
#define GG_DBGLVL_INFO          (1 << 2)

DebugLogMaskInfo gGgLogLevels[] = {
    {
        .Mask = GG_DBGLVL_INFO,
        .Name = "info",
        .Description = "Basic information on high level actions"
    },
    {
        .Mask = GG_DBGLVL_TYPE,
        .Name = "type",
        .Description = "Information on packet types + sizes"
    },
    {
        .Mask = GG_DBGLVL_PKTDATA,
        .Name = "pktdata",
        .Description = "Packet data"
    }
};

ModuleLoggingInfo gGgLoggingInfo = {
    .ModuleName = "gasgauge",
    .MaskList = gGgLogLevels,
    .NumLevels = count_of(gGgLogLevels),

    .GlobalDebugLevels = {
        GG_DBGLVL_INFO
    },
};

EFI_DRIVER_ENTRY_POINT(InstallGasGauge)

EFI_STATUS
InstallGasGauge (
        IN EFI_HANDLE       ImageHandle,
        IN EFI_SYSTEM_TABLE *SystemTable
        )
{
    EFI_STATUS Status;
    ConsoleSourceRef *RefKey;

    EfiInitializeDriverLib(ImageHandle, SystemTable);
    EFI_DEVICE_PATH_PROTOCOL *Path = NULL;

    Status = EfiCoreLogRegisterModule(&gGgLoggingInfo, &RefKey, TRUE, FALSE);
    RETURN_EFI_ERROR(efiStatus);
}
```

In this example, the gas-gauge driver is registering a source (named *gasgauge*) to be the default source for this driver. In the call to `EfiCoreLogRegisterModule`, it also signals that it is not a library.

2.5 Multiple Sources

Some drivers (like *PmuCore*) are multi-function drivers, i.e they install multiple protocols that represent different functionalities (like *Charger*, *Adc* and *PowerSource*). Drivers like these can register multiple sources - the only consideration is to ensure that the `ConsoleSourceRef` keys passed back by the framework be stored so that they can be passed into the various logging functions:

```
ModuleLoggingInfo gCharger = {
    .ModuleName = "charger",
    .LevelList = gChargerLevels,
    .NumLevels = count_of(gChargerLevels),

    .GlobalDebugLevels = {
        PMU_CHARGER_INFO
    },
};

ModuleLoggingInfo gAdc = {
    .ModuleName = "adc",
    .LevelList = gAdcLevels,
    .NumLevels = count_of(gAdcLevels),

    .GlobalDebugLevels = {
        PMU_ADC_INFO
    },
};

EFI_DRIVER_ENTRY_POINT(InstallGasGauge)

ConsoleSourceRef gAdcRef = NULL_CONSOLE_REF;
ConsoleSourceRef gChargerRef = NULL_CONSOLE_REF;

EFI_STATUS
InstallPmuCore (
        IN EFI_HANDLE       ImageHandle,
        IN EFI_SYSTEM_TABLE *SystemTable
        )
{
    EFI_STATUS Status;

    Status = EfiCoreLogRegisterModule(&gCharger, &gChargerRef, FALSE, FALSE);
    RETURN_EFI_ERROR(efiStatus);

    Status = EfiCoreLogRegisterModule(&gAdc, &gAdcRef, FALSE, FALSE);
    RETURN_EFI_ERROR(efiStatus);
}
```

See section 3.2 *API* for an example of using the provided `ConsoleSourceRef` in a logging function.

2.6 Libraries

When registering from a library, it's recommended that a method be exposed to allow the consumer to explicitly control registration of the logging source.

In addition, the `ConsoleSourceRef` key should be made static to the file to prevent name collisions with other libraries.

Finally, a value of *TRUE* should be passed to the `IsLibrary` argument in the `EfiCoreLogRegisterModule` function.

```
STATIC ModuleLoggingInfo gAudioLib = {
    .ModuleName = "audiolib",
    .LevelList = gAudioLibLevels,
    .NumLevels = count_of(gChargerLevels),

    .GlobalDebugLevels = {
        AUDIOLIB_INFO
    },
};

STATIC EntityRef gAudioLibRef = NULL;

EFI_STATUS
AudioLib_RegisterLogging (
    )
{
    return EfiCoreLogRegisterModule(&gAudioLib, &gAudioLibRef, FALSE, TRUE);
}
```

NOTE: Do not register libraries as the default source for the image, as all prints from that image will be tagged with the in-correct source.

2.7 Global Debug

Since global debug prints get logged into the forensics buffer, careful consideration has to be given to the the assignment of masks to the `GlobalDebugLevels[0]` field to ensure no performance drop-offs when adding new prints (logging to forensics does take a small but finite amount of time).

In general, a source should only map high-level, informational prints that provide some level of visibility into driver operations to the global debug level.

Intensive prints like dumping packet contents, or listing out explicit transaction details in a detailed manner **do not** belong to the global debug level. It's generally recommended to register an 'info' debug mask that encapsulates this high-level of debug verbosity.

3 Logging

This section goes over the new logging methods as well as providing guidelines for legacy methods.

3.1 Visibility

Unlike previous methods like `LogStatus` or `PrintBlock` that required you to include additional libraries, the new logging methods are visibile from any-where in EFI (not PreEFI however). The only pre-requisite for usage is the inclusion of "EfiDriverLib.h" in your C source as well as linking against EfiDriverLib. Since almost all drivers and libraries already do this, it should amount to little work for most developers.

3.2 API

Default Source:

All of the below methods get tagged with the default source for that image. For legacy sources, this means that the source name will be set to the image name. For registered sources, it will bind to the source that was registered as default. If sources were registered but none were set to default, the tagging will default to selecting the legacy source for that image (which is registered automatically in the call to `EfiInitializeDriverLib`).

Method	Description
`AsciiPrint(Format, ...)`	Outputs 'print' type text
`LOG_ERROR(Format, ...)`	Outputs 'error' type text
`LOG_WARN(Format, ...)`	Outputs 'warn' type text
`LOG_DEBUG(Mask, Format, ...)`	Outputs 'debug' type text given a debug mask
`LOG_DUMP(Mask, Buffer, Len)`	Dumps out the contents of a buffer in hex over the 'debug' text type
`LOG_TRACE_FUNCENTRY()`	Marks entry into a function, of 'trace' type
`LOG_TRACE_FUNCEXIT()`	Marks exit out of a function, of 'trace' type
`LOG_TRACE_MARKER(Format, ...)`	Custom marker break-crumbs, outputs 'trace' type

NOTE: A call to `LOG_DEBUG` only works if a default source that has debug-masks has been registered by the image from which that call is being made.

Specific Source:

To tag a print as emnating from something other than the default source, just call the above functions with an 'M' pre-pended to the function name (e.g `MLOG_ERROR`, `MAsciiPrint` or `MLOG_DEBUG`).

All of the 'M' series functions require the user pass in the `ConsoleSourceRef` key as an additional first argument - this is especially pertinent for multi-function drivers and libraries:

```
MLOG_ERROR(Ref, Format, ...)
MLOG_DEBUG(Ref, Mask, Format, ...)
```

Sample code:

```
#define AUDIOLIB_INFO    (1 << 0)

static ConsoleSourceRef gAudioLibRef = NULL_CONSOLE_REF;

void
SomeFunction() (
    )
{
    MAsciiPrint(gAudioLefRef, "Test print!\n");
    MLOG_DEBUG(gAudioLibRef, AUDIOLIB_INFO, "Test info debug print!\n");
}

static ModuleLoggingInfo gAudioLib = {
    .ModuleName = "audiolib",
    .MaskList = gAudioLibLevels,
    .NumMasks = count_of(gChargerLevels),

    .GlobalDebugLevels = {
        AUDIOLIB_INFO
    },
};

EFI_STATUS
AudioLib_RegisterLogging (
    )
{
    return EfiCoreLogRegisterModule(&gAudioLib, &gAudioLibRef, FALSE, TRUE);
}
```

3.3 Legacy Functionality

All legacy functions (like `DEBUG` and `LogDebug`) have been ported over to the new architecture. However, it is ***not recommended*** to use these functions anymore on a continuing basis. All new drivers should register sources and utilize the new API for their logging needs.

3.3.1. DEBUG() and LogDebug()

Since both `LOG_DEBUG` and `MLOG_DEBUG` calls require the caller to pass in a debug-mask associated with the default/specific module respectively, there was no easy way to map over the `DEBUG` and `LogDebug` calls.

Hence, a new *system* source was created to house the `DEBUG` print family as well as `LogDebug`. This source is available system-wide and can be seen by any entity that has access to gRT.

All of the `EFI_D_*` defines have been mapped under the system source:

```
DebugLogMaskInfo gSystemLevelList[] = {
    {
        .Mask = EFI_D_INIT,
        .Name = "init",
        .Description = "Initialization prints"
    },
    {
        .Mask = EFI_D_WARN,
        .Name = "warn",
        .Description = "Warnings"
    },
    {
        .Mask = EFI_D_LOAD,
        .Name = "load",
        .Description = "Driver load"
    },
    {
        .Mask = EFI_D_ERROR,
        .Name = "error",
        .Description = "Error/info",
    },
    {
        .Mask = EFI_D_DEBUG,
        .Name = "debug",
        .Description = "LogDebug() prints"
    },
    ...
    ...
};

ModuleLoggingInfo gSystemLoggingInfo = {
    .ModuleName = "system",
    .MaskList = gSystemMaskList,
    .NumMasks = count_of(gSystemMaskList),
    .GlobalDebugLevels = {
        EFI_D_ERROR,
    }
};
```

A new `EFI_D_DEBUG` define has been added under the *system* source, and `LogDebug` has been mapped to that.

3.3.2. Other functions

`LogStatus` has been mapped to `AsciiPrint`, while `LogError` has been mapped to `LOG_ERROR`.

4 Summary

4.1 Guidelines

1. When writing a new driver, please refrain from using any of the legacy methods. Rather, create a new source as well as appropriate debug flags and use those instead.
2. For debug spew, do not use `DEBUG` or `LogDebug` going forward (both new and old drivers). Rather, use the new `LOG_DEBUG` macro instead with custom debug masks
3. Be very mindful of the masks you assign to the `GlobalDebugMask` field since they get logged to forensics and can affect performance (See 2.7 *Global Debug*).

4.2 Sample Drivers

There are a few sample drivers that showcase correct usage of the new framework methods. These are:

Drivers:

1. **Gas Gauge -** bootloader/Platform/TI/Common/Chipset/BQ27541
2. **UART Swif -** bootloader/Platform/Samsung/Common/Chipset/S5L8940/UartSwif
3. **Serial -** bootloader/Platform/Samsung/Common/Chipset/Common/Uart

Libraries:

1. **I2clib (for libraries) -** bootloader/Platform/Apple/Common/Library/I2CLib
2. **Ezlink (for libraries) -** bootloader/Platform/Apple/Common/Library/EzLink