안드로이드 OpenCore 멀티미디어 프레임워크

게시 : 2009년 5월 13일

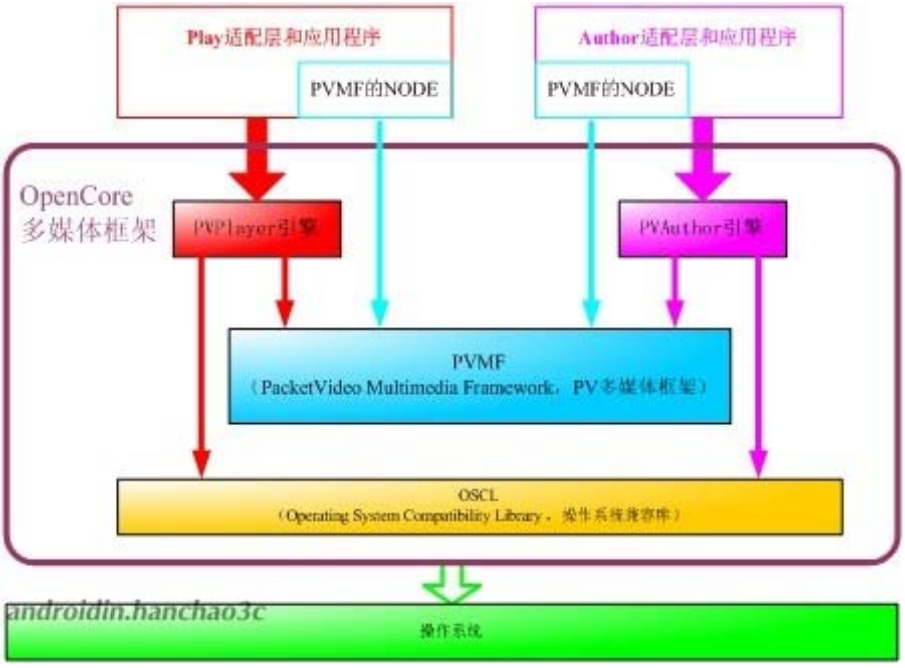
번역 및 약간의 수정 : 고도리(http://www.aesop.or.kr), 2009/08/13

OpenCore 개요

Opencore는 안드로이드의 멀티미디어 프레임워크로 다른 이름으로는 일반적으로 PacketVideo라고 불린다. 다른 이름으로 PacketVideo는 회사이름이기도 하고, 멀티미디어 framework는 software layer의 이름이기도 하다.
두 개의 개념가운데에 있는 안드로이드 개발자 입장에서는 둘 다 동일한 개념이다. 안드로이드를 다른 라이브러리와 비교했을때 OpenCore 코드는 매우 양이 많고, C++로 작성된 full-featured(전체적인 멀티미디어 기능을 갖는) 운영체제에 통합되는 구조로 되어 있고(operating system migration layer), 매우 다양한 함수들이 상속성의 특징과 같은 다양한 레벨간의 인터페이스 여러형태로 패키징 되어 있다.
OpenCore 멀티미디어 프레임 워크를 거시적인 관점에서 볼 때, 그것은 주로 두 가지 측면을 포함하고있다 :

PVPlayer: 다양한 오디오 비디오 스트림에 대한 재생 기능을 갖고 있는 미디어 플레이어를 위한 함수들을 제공
PVAuthor : 오디오, 비디오 스트림을 녹화하고 이와 더불어 이미지 캡쳐 기능을 위한 함수들을 제공
PVPlayer 및 PVAuthor는 개발자들이 사용할 수 있는 형태로 SDK를 제공하며, 다양한 응용프로그램과 서비스를 SDK를 이용하여 작성할 수 있다. 자주 모바일 단말기에 멀티미디어에서 미디어 플레이어, 카메라, 비디오 레코더, 테이프 레코더와 같은 애플 리케이션에 사용된다.
전반적인 구조를 이해하기 위하여 거시적인 관점에서의 OpenCore software는 여러개의 레벨로 나눌 수 있다.

宏观: 거시적, macro


Play适配层和应用程序
PVMF的NODE
Author适配层和应用程序
PVMF的NODE
OpenCore
多媒体框架
PVPlayer引擎
PVAuthor引擎
PVMF
（PacketVideo Multimedia Framework，PV多媒体框架）
OSCL
（Operating System Compatibility Library，操作系统兼容库）
androidin.hanchao3c
操作系统

OSCL : 운영 체제 호환성 라이브러리 (Operating System Compatibility Library), 더 좋은 다른 운영체제간의 호환성을 위하여 기본 운영체제 동작을 지원하는 기능을 포함하고 있다. 기본 데이터 형식, configuration, string instruments, IO, error handling, thread 등을 포함한 C++ 기본 라이브러리와 유사하다.

PVMF : PacketVideo 멀티미디어 프레임 워크 (PV multimedia framework), document analysis (parser)와 composition (composer)를 구현한 framework이고, 이 안의 codec NODE는 공통적인 인터페이스를 상속할 수 있고 , 사용자 계층은 NODE를 생성하기 위하여 그 공통 인터페이스를 상속 할 수 있다.

PVPlayer 엔진 : PVPlayer 엔진.
PVAuthor 엔진 : PVAuthor 엔진.

사실 OpenCore의 내용은 아주 넓은 내용을 포함하고 있다:
player의 입장에서 PVPlayer는 입력(Source)으로 network file 혹은 media stream 등이 될 수 있고, 출력(Sink)은 오디오/비디오 장비의 입력이 될 수 있고, 기본적인 기능을 포함하는 미디어 흐름 제어와 document analysis, video streaming, audio decoder(Decode)와 그 외의 다른 특징을 갖고 있다. paper document서부터 방송 미디어까지 포함하고 있고, 또한 network-related RTSP streaming 기능도 포함하고 있다.

media area에서의 recording에 관련해서, PVAuthor의 입력(Source)으로는 카메라, 마이크, 다른 장비가 될 수 있고, 그 출력(Sink)는 video streaming, audio encoding(Encode) 등의 다양한 형태를 갖게 된다.

미디어 영역의 recording에 있어서, PVAuthor 입력 (원본)은 카메라, 마이크 및 기타 장비, 출력 (싱크) 각종 문서의 동기화의 흐름, 비디오 (인코딩)로 작성된 문서 등 오디오 인코딩 스트리밍 등 기능을 수행합니다.

OpenCore SDK의 사용에 있어, 응용프로그램 계층에서 adaptor(Adaptor)를 구현하는 것이 필요하고, 그 adaptor는 PVMF를 위한 NODE의 특정기능을 common interface를 이용해서 구현해야하는데 이것은 상위단에서의 사용을 위하여 plug-in 형태로 구현하게 된다.

2.1 코드 구조
OpenCore의 코드를 보려면 /external/opencore 디렉토리의 서브디렉토리를 보면 된다.

android: android를 위한 SDK의 최상위 base 코드가 들어있다. PVPlayer와 PVAuthor에 그 기초를 두고 있으며, player와 author의 사용에 대한 코드가 들어있다.
baselibs: data structure와 thread-safe한 기본 library들, util들
codecs_v2: codec들(audio/video), 코덱에 대한 OpenMAX plug-in들, utility들(칼라변환,

parser등)
engines: PVAuthor, PVPlayer 등의 엔진 소스들
extern_libs_v2: khronos란 디렉토리에 OpenMAX header file들이 들어 있다.
fileformats: Analysis of file formats (parser) Tools ==> 실제로는 engine test코드에서만 쓰이는 듯 하다(ex> avi등의 경우)
nodes: PVMF가 제공하는 NODE에 대한 코드가 있다. 주로 codec과 parser등에 대한 node code가 있다.
oscl: operating system-compatible library
pvmi: PVMF의 input, output control interface에 대한 코드가 있다.
protocols: RTSP, RTP, HTTP등의 네트워크 관련 코드들이 들어 있다.
pvcommon: pvcommon library를 만드는 Android.mk file만 있음 여기에는 소스는 없다. 즉, 다른 디렉토리의 라이브러리들을 모아서 라이브러리로 만드는 makefile이 있을 뿐이다.
pvplayer: pvplayer library를 만드는 Android.mk file만 있음 여기에는 소스는 없다. 즉, 다른 디렉토리의 라이브러리들을 모아서 라이브러리로 만드는 makefile이 있을 뿐이다.
pvauthor: pvauthor library를 만드는 Android.mk file만 있음 여기에는 소스는 없다. 즉, 다른 디렉토리의 라이브러리들을 모아서 라이브러리로 만드는 makefile이 있을 뿐이다.
tools_v2: compiler tool들과 등록될 수 있는 몇 개의 module들이 있다.

external/opencore 디렉토리에는 다음과 같은 두 개의 파일이 존재한다.

Android.mk: compile the overall document
pvplayer.conf: configuration file

external/opencore 디렉토리의 모든 sub 디렉토리에는 많은 Android.mk 파일들이 있고, 그것들 사이에는 서로 “재귀적”인 관계에 있다(즉, 서로가 서로를 포함하는관계),

예를 들어 external/opencore/Android.mk 파일에는

CODE:
include $(PV_TOP)/pvcommon/Android.mk
include $(PV_TOP)/pvplayer/Android.mk
include $(PV_TOP)/pvauthor/Android.mk
...

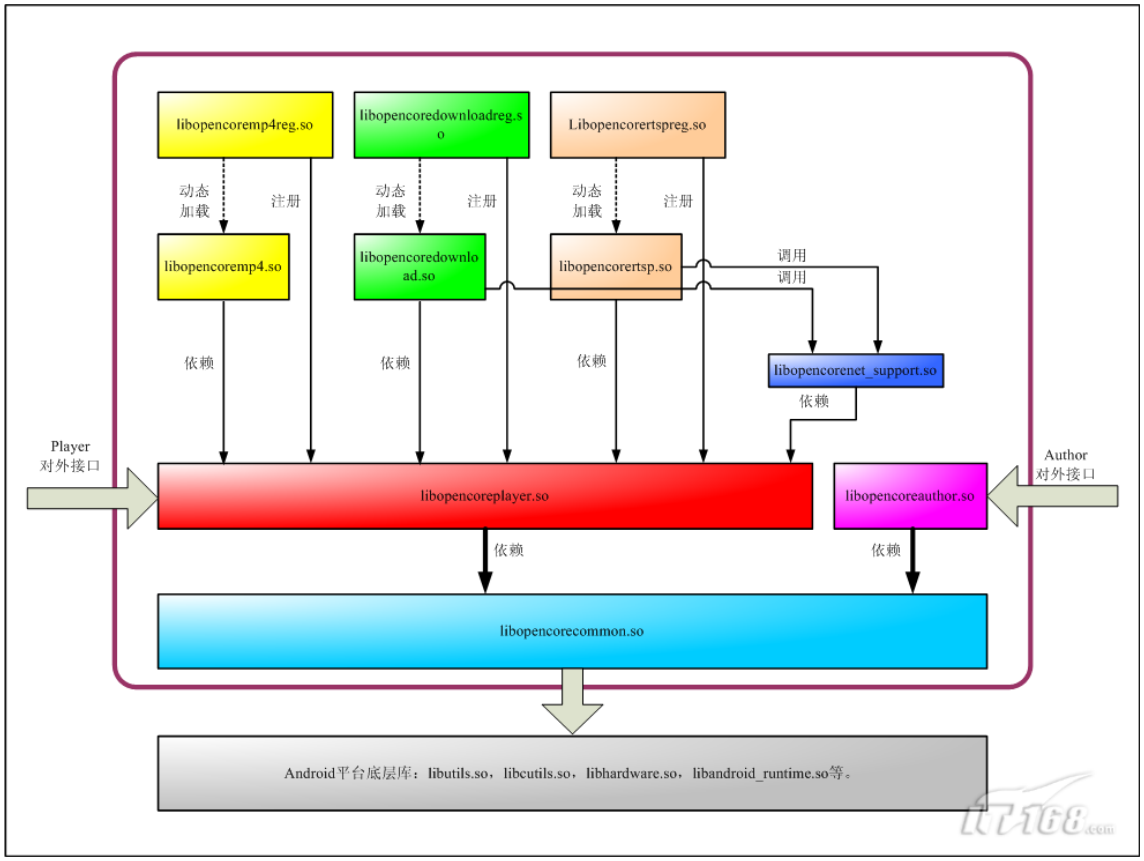
등의 내용이 있다.

즉, 각각의 Android.mk 파일에서는 서로 서로를 호출하는 구조로 되어 있다.

2.2 Compile 구조
컴파일 된 각종 라이브러리들은 다음과 같다.

libopencoreauthor.so : OpenCore author library
libopencorecommon.so : OpenCore 기본 common library
libopencoredownloadreg.so : Download the registration database
libopencoredownload.so : download 관련 라이브러리
libopencoremp4reg.so : MP4 레지스트리
libopencoremp4.so : MP4 함수관련
libopencorenet_support.so : 네트워크 지원 라이브러리
libopencoreplayer.so : OpenCore player library
libopencorertspreg.so : RTSP 레지스트리
libopencorertsp.so : RTSP 함수 관련 라이브러리

위의 라이브러리 들에 대한 계층도는 다음과 같다.



注册: 등록하다. registration

OpenCore의 다양한 라이브러리들에는 다음과 같은 관계가 있다.

libopencorecommon.so: 다양한 기본 라이브러리를 제공
libopencoreplayer.so 와 libopencoreauthor.so: 각각 playback과 recording에 사용이 되고, 이 라이브러리들은 외부 인터페이스를 담당하게 된다.

libopencorenet_support.so 는 네트워크를 지원하는 함수들을 제공한다
많은 형태의 다양한 plug-in(Plug-in) 방식의 player를 추가하는 방법으로는 두 개의 함수가 사용이 되는데, 하나는 등록에 사용이 되고, 다른 하나는 실제 구현코드가 들어있다.

2.2. libopencorecommon.so 라이브러리 구조

libopencorecommon.so 라이브러리는 전체 OpenCore library의 핵심이고, compile은 pvcommon/Android.mk 파일을 이용해서 하게 된다. 이 mk 파일에는 다른 sub 디렉토리에 대한 mk 파일을 호출하는 내용을 볼 수 있다.

CODE:
```
include $(BUILD_SHARED_LIBRARY)
include $(PV_TOP)//oscl/oscl/osclbase/Android.mk
include $(PV_TOP)//oscl/oscl/osclerror/Android.mk
include $(PV_TOP)//oscl/oscl/osclmemory/Android.mk
include $(PV_TOP)//oscl/oscl/osclutil/Android.mk
include $(PV_TOP)//oscl/pvlogger/Android.mk
include $(PV_TOP)//oscl/oscl/osclproc/Android.mk
include $(PV_TOP)//oscl/oscl/osclio/Android.mk
include $(PV_TOP)//oscl/oscl/osclregcli/Android.mk
include $(PV_TOP)//oscl/oscl/osclregserv/Android.mk
include $(PV_TOP)//oscl/unit_test/Android.mk
include $(PV_TOP)//oscl/oscl/osclib/Android.mk
include $(PV_TOP)//pvmi/pvmf/Android.mk
include $(PV_TOP)//baselibs/pv_mime_utils/Android.mk
include $(PV_TOP)//nodes/pvfileoutputnode/Android.mk
include $(PV_TOP)//baselibs/media_data_structures/Android.mk
include $(PV_TOP)//baselibs/threadsafe_callback_ao/Android.mk
include $(PV_TOP)//codecs_v2/utilities/colorconvert/Android.mk
include $(PV_TOP)//codecs_v2/audio/gsm_amr/amr_nb/common/Android.mk
include $(PV_TOP)//codecs_v2/video/avc_h264/common/Android.mk
```

이 makefile에는 실제로 컴파일 되어야 하는 다른 디렉토리에 대한 makefile에 대한 리스트를 가지고 있다.
libopencorecommon.so 라이브러리에는 다음과 같은 내용이 포함되어 있다.
■ OSCL
■ framework 중 PVMF 관련 내용 (pvmi / pvmf / Android.mk)
■ 기본적인 라이브러리 (baselibs)

■ codec 관련 내용
■ document output node (nodes / pvfileoutputnode / Android.mk)

이 라이브러리는 밑에는 OSCL을 가지고 있고, 이와 마찬가지로 Node tool과 codec을 지원하는 PVMF framework에 대한 기능도 가지고 있다.

2.3. libopencoreplayer.so 라이브러리 구조

pvplayer/Android.mk 를 보면 다음과 같이 되어 있다.

CODE:
```
include $(BUILD_SHARED_LIBRARY)
include $(PV_TOP)//engines/player/Android.mk
include $(PV_TOP)//codecs_v2/audio/aac/dec/util/getactualaacconfig/Android.mk
include $(PV_TOP)//codecs_v2/video/avc_h264/dec/Android.mk
include $(PV_TOP)//codecs_v2/audio/aac/dec/Android.mk
include $(PV_TOP)//codecs_v2/audio/gsm_amr/amr_nb/dec/Android.mk
include $(PV_TOP)//codecs_v2/audio/gsm_amr/amr_wb/dec/Android.mk
include $(PV_TOP)//codecs_v2/audio/gsm_amr/common/dec/Android.mk
include $(PV_TOP)//codecs_v2/audio/mp3/dec/Android.mk
include $(PV_TOP)//codecs_v2/utilities/m4v_config_parser/Android.mk
include $(PV_TOP)//codecs_v2/utilities/pv_video_config_parser/Android.mk
include $(PV_TOP)//codecs_v2/omx/omx_common/Android.mk
include $(PV_TOP)//codecs_v2/omx/omx_queue/Android.mk
include $(PV_TOP)//codecs_v2/omx/omx_h264/Android.mk
include $(PV_TOP)//codecs_v2/omx/omx_aac/Android.mk
include $(PV_TOP)//codecs_v2/omx/omx_amr/Android.mk
include $(PV_TOP)//codecs_v2/omx/omx_mp3/Android.mk
include $(PV_TOP)//codecs_v2/omx/factories/omx_m4v_factory/Android.mk
include $(PV_TOP)//codecs_v2/omx/omx_proxy/Android.mk
include $(PV_TOP)//nodes/common/Android.mk
include $(PV_TOP)//pvmi/content_policy_manager/Android.mk
include $(PV_TOP)//pvmi/content_policy_manager/plugins/oma1/passthru/Android.mk
include $(PV_TOP)//pvmi/content_policy_manager/plugins/common/Android.mk
include $(PV_TOP)//pvmi/media_io/pvmiofileoutput/Android.mk
include $(PV_TOP)//fileformats/common/parser/Android.mk
include $(PV_TOP)//fileformats/id3parcom/Android.mk
include $(PV_TOP)//fileformats/rawgsmamr/parser/Android.mk
include $(PV_TOP)//fileformats/mp3/parser/Android.mk
include $(PV_TOP)//fileformats/mp4/parser/Android.mk
```

include $(PV_TOP)//fileformats/rawaac/parser/Android.mk
include $(PV_TOP)//fileformats/wav/parser/Android.mk
include $(PV_TOP)//nodes/pvaacffparsernode/Android.mk
include $(PV_TOP)//nodes/pvmp3ffparsernode/Android.mk
include $(PV_TOP)//nodes/pvamrffparsernode/Android.mk
include $(PV_TOP)//nodes/pvmediaoutputnode/Android.mk
include $(PV_TOP)//nodes/pvomxvideodecnode/Android.mk
include $(PV_TOP)//nodes/pvomxaudiodecnode/Android.mk
include $(PV_TOP)//nodes/pvwavffparsernode/Android.mk
include $(PV_TOP)//pvmi/recognizer/Android.mk
include $(PV_TOP)//pvmi/recognizer/plugins/pvamrffrecognizer/Android.mk
include $(PV_TOP)//pvmi/recognizer/plugins/pvmp3ffrecognizer/Android.mk
include $(PV_TOP)//pvmi/recognizer/plugins/pvwavffrecognizer/Android.mk
include $(PV_TOP)//engines/common/Android.mk
include $(PV_TOP)//engines/adapters/player/framemetadatautility/Android.mk
include $(PV_TOP)//protocols/rtp_payload_parser/util/Android.mk
include $(PV_TOP)//android/Android.mk
include $(PV_TOP)//android/drm/oma1/Android.mk
include $(PV_TOP)//tools_v2/build/modules/linux_rtsp/core/Android.mk
include $(PV_TOP)//tools_v2/build/modules/linux_rtsp/node_registry/Android.mk
include $(PV_TOP)//tools_v2/build/modules/linux_net_support/core/Android.mk
include $(PV_TOP)//tools_v2/build/modules/linux_download/core/Android.mk
include $(PV_TOP)//tools_v2/build/modules/linux_download/node_registry/Android.mk
include $(PV_TOP)//tools_v2/build/modules/linux_mp4/core/Android.mk
include $(PV_TOP)//tools_v2/build/modules/linux_mp4/node_registry/Android.mk

libopencoreplayer.so 는 다음과 같은 것을 포함하고 있다.

■ decoding tool
■ file parser(mp4)
■ Node corresponding decoding tool
■ player engine part (engines / player / Android.mk)
■ Android's player adapter for the (android / Android.mk)
■ identification tools (pvmi / recognizer)
■ codec part OpenMax tools (codecs_v2/omx)
■ Node corresponding to the registration of a number of plug-ins

libopencoreplayer.so의 주요 내용은 각 parser와 decoder 관련 내용들이다.
PVPlayer는 engines/player/Android.mk 에 그 핵심 함수들이 있고, android/Android.mk 에는 player로 사용하기 위하여 PVPlayer를 Android에 built-in 시킨 코드가 있다.

2.4. libopencoreauthor.so 라이브러리 구조 분석
recording관련 media library인 libopencoreauthor.so에 대한 makefile은 pvauthor/Android.mk 파일이다.

CODE:
include $(BUILD_SHARED_LIBRARY)
include $(PV_TOP)//engines/author/Android.mk
include $(PV_TOP)//codecs_v2/video/m4v_h263/enc/Android.mk
include $(PV_TOP)//codecs_v2/audio/gsm_amr/amr_nb/enc/Android.mk
include $(PV_TOP)//codecs_v2/video/avc_h264/enc/Android.mk
include $(PV_TOP)//fileformats/mp4/composer/Android.mk
include $(PV_TOP)//nodes/pvamrencnode/Android.mk
include $(PV_TOP)//nodes/pvmp4ffcomposernode/Android.mk
include $(PV_TOP)//nodes/pvvideoencnode/Android.mk
include $(PV_TOP)//nodes/pvavcencnode/Android.mk
include $(PV_TOP)//nodes/pvmediainputnode/Android.mk
include $(PV_TOP)//android/author/Android.mk

libopencoreauthor.so 는 다음과 같은 기능을 갖고 있다.
■ encoding tools (video streaming H263, H264, audio stream Amr)
■ the composition of device files (mp4)
■ Node corresponding encoding tools
■ media input Node (nodes / pvmediainputnode / Android.m)
■ author part of the engine (engines / author / Android.mk)
■ the author for Android adapter (android / author / Android.mk)

libopencoreauthor.so 라이브러리는 각 component에 대한 encoder와 muxer가 있다.

PVAuthor는 engines/author/Android.mk 에 그 핵심 함수들이 있고, android/Android.mk 에는 media recorder로 사용하기 위하여 PVAuthor를 Android에 built-in 시킨 코드가 있다.

2.5 그 외 라이브러리

그 외 external/opencore/Android.mk file 에는 다음과 같은 내용이 들어 있다:

Network support library libopencorenet_support.so:
tools_v2/build/modules/linux_net_support/core/Android.mk

MP4 functions library인 libopencoremp4.so and registration library libopencoremp4reg.so:
tools_v2/build/modules/linux_mp4/core/Android.mk
tools_v2/build/modules/linux_mp4/node_registry/Android.mk

RTSP functions library libopencorertsp.so and registration library libopencorertspreg.so:
tools_v2/build/modules/linux_rtsp/core/Android.mk
tools_v2/build/modules/linux_rtsp/node_registry/Android.mk

Download 관련 library libopencoredownload.so and registration library libopencoredownloadreg.so:
tools_v2/build/modules/linux_download/core/Android.mk
tools_v2/build/modules/linux_download/node_registry/Android.mk

3. OSCL

OSCL(operating system-compatible library)은 다양한 운영체제에 대해 포팅이 가능하도록 구성되어 있고, 그 code structure는 다음과 간다.

```
oscl/oscl
|-- config : configuration macro
|-- makefile
|-- makefile.pv
|-- osclbase :  기본적인 데이터 타입, macro, STL container와 비슷한 함수들이 존재
|-- osclerror : error-handling functions
|-- osclio : file IO and Socket functions
|-- oscllib : dynamic library interface function
|-- osclmemory  :  memory management(auto pointer등과 같은 기능 포함)
|-- osclproc : thread, multi-task communications and other functions
|-- osclregcli  : client 등록에 대한 함수
|-- osclregserv : 서버 등록관련 함수들
`-- osclutil  : string처리와 같은 기본 유틸리티 함수들
```

oscl 디렉토리는 각 모듈별 디렉토리로 나누어져 있다. OSCL에는 관련된 함수들이 아주 자세하게 설명되고 있으며, C 언어를 주로 사용하며, 상윗단에서의 사용에 있어서는 C++ 인터페이스가 사용이 된다. 사실 OpenCore에서의 PVMF의 사용은 OSCL에 의존적이며, 전체 OpenCore에 대한 호출은 OSCL부분을 사용하는게 필요하다.

OSCL의 구현에 있어 많은 전형적인 C 함수는 간단한 패키지의 형태로 존재하게 된다. 예를 들어 수학함수와 관련된 osclutil의 해당 부분은 inline 함수의 형태로 oscl_mach.inl 파일에 존재하게 된다.

CODE:
```
OSCL_COND_EXPORT_REF OSCL_INLINE double oscl_log(double value)
{
return (double) log(value);
}
OSCL_COND_EXPORT_REF OSCL_INLINE double oscl_log10(double value)
{
return (double) log10(value);
}
OSCL_COND_EXPORT_REF OSCL_INLINE double oscl_sqrt(double value)
{
return (double) sqrt(value);
}
```

oscl_mach.inl 파일은 oscl_math.h에 include되고 결과적으로 oscl_log()함수는 log()함수를 사용한 것과 동일하게 된다.
C 언어 스탠다드 라이브러리는 C++의 형태로 정의가 되지만, 그렇게 복잡하지는 않은 형태이다. 예를 들어서 oscllib 디레토리는 다음과 같은 구조를 가지고 있다.

```
oscl/oscl/oscllib/
|-- Android.mk
|-- build
| `-- make
| `-- makefile
`-- src
|-- oscl_library_common.h
|-- oscl_library_list.cpp
|-- oscl_library_list.h
|-- oscl_shared_lib_interface.h
|-- oscl_shared_library.cpp
`-- oscl_shared_library.h
```

위의 코드에 있는 oscl_shared_library.h 는 다음과 같이 선언되어 있다.

CODE:
```
class OsclSharedLibrary
{
```

```
public:
OSCL_IMPORT_REF OsclSharedLibrary();
OSCL_IMPORT_REF OsclSharedLibrary(const OSCL_String& aPath);
OSCL_IMPORT_REF ~OsclSharedLibrary();
OSCL_IMPORT_REF OsclLibStatus LoadLib(const OSCL_String& aPath);
OSCL_IMPORT_REF OsclLibStatus LoadLib();
OSCL_IMPORT_REF void SetLibPath(const OSCL_String& aPath);
OSCL_IMPORT_REF OsclLibStatus QueryInterface(const OsclUuid& aInterfaceId,
OsclAny*& aInterfacePtr);
OSCL_IMPORT_REF OsclLibStatus Close();
OSCL_IMPORT_REF void AddRef();
OSCL_IMPORT_REF void RemoveRef();
}
```

oscl_shared_library.cpp에 있는 이런 코드는 라이브러리를 로딩하는데 사용되는 코드이며, 이러한 것들은 동적으로 라이브러리를 로딩하기 위하여 dlopen() 함수등을 사용한다.

4. Codec 및 Fileformat

멀티미디어는 file format처리와 codec 이 두 개에 그 기본을 둔다. 그리고, 멀티미디어 application은 broadcast media(PlayBack)과 media recording을 작성하는 것이 대부분이다.
media player 프로세스에서 일반적으로 player는 media file을 처리함에 있어 두 개의 과정으로 처리하게 되는데 하나는 file에 대한 parsing이고, 나머지는 media stream에 대한 decoding이다.
예를 들어, mp4 file을 처리함에 있어, 여기서 이 mp4 파일은 AMR 혹은 AAC audio stream을 갖고 있고, video stream으로는 H263, MPEG4, AVC(H264)를 갖을 수 있다. 이 stream들은 3GP package의 형태로 packaging되어 있게 된다.
media player는 packaging된 형태로부터 각 stream을 배낸 후, media stream들을 decoding하게 된다.

media 처리에서 recording은 video, audio, image 캡쳐 기능과 직접 연관되어 있다. video와 오디오를 하나의 파일로 저장하는 기능은 정확히 player의 반대의 기능이고, 하드웨어 기기로부터 video와 audio atream을 획득하고, 이것을 코딩한 후 적절한 형태로 저장을 하게 된다(file의 형태건 stream의 형태건).
OpenCore는 이 두 개의 파트를 fileformats 과 codecs_v2라는 디렉토리로 각각 분리해 놓았다. 이러한 두 개의 부분은 아주 기본적인 기능들이며, 다른 모듈에 의해서 호출된다. Node의 구성에서와 같은 모듈에서.....호출된다.

4.1 file format 처리

play와 recording시에는 두가지 형태의 file을 다루는 방법이 존재하게 되는데, OpenCore에서 파일 포맷을 다루는 두 개의 타입은 parser와 composer(component device)라고 부른다.(demuxer와 muxer라고 보면 되겠다)

fileformats 디렉토리는 다음과 같이 구성되어 있다.

```
fileformats
|-- avi
|   `-- parser
|-- common
|   `-- parser
|-- id3parcom
|   |-- Android.mk
|   |-- build
|   |-- include
|   `-- src
|-- mp3
|   `-- parser
|-- mp4
|   |-- composer
|   `-- parser
|-- rawaac
|   `-- parser
|-- rawgsmamr
|   `-- parser
`-- wav
    `-- parser
```

각 디렉토리들은 다양한 file format에 대한 코드가 들어 있다.

4.2 codec

codecs_v2 디렉토리에는 각종 코덱에 대한 코드들이 들어 있다.

codecs_v2 디렉토리의 구조는 다음과 같다.

```
codecs_v2
|-- audio
|   |-- aac
```

```
|   |-- gsm_amr
|   |-- mp3
|   `-- sbc
|-- omx
|   |-- factories
|   |-- omx_aac
|   |-- omx_amr
|   |-- omx_common
|   |-- omx_h264
|   |-- omx_m4v
|   |-- omx_mp3
|   |-- omx_proxy
|   `-- omx_queue
|-- utilities
|   |-- colorconvert
|   |-- m4v_config_parser
|   `-- pv_video_config_parser
`-- video
    |-- avc_h264
    `-- m4v_h263
```

audio, video 디렉토리에는 각 코덱에 대한 디렉토리가 존재하고, 그 서브디렉토리에는 dec, enc 와 같은 디코딩, 인코딩 코드가 들어있다.

```
`-- video
    |-- avc_h264
    |   |-- common
    |   |-- dec
    |   |-- enc
    |   `-- patent_disclaimer.txt
    `-- m4v_h263
        |-- dec
        |-- enc
        `-- patent_disclaimer.txt
```

codecs_v2 디렉토리는 omx 라는 디렉토리를 가지고 있으며, 이 디렉토리에는 khronos 그룹의 OpenMAX 함수들이 들어 있다. OpenMAX는 2006년도에 시작된 NVIDIA에 의한 khronos그룹의 멀티미디어 어플리케이션 standard에 대한 framwork이다.

OpenMax는 3개의 Layer로 나뉘어 지며,

OpenMax AL : 최상위 Level의 Application Layer로, platform에 무관하게 media interface를 제공하고 사용할 수 있다. 다만 문제는 고수준 미디어 제어 프로그램만을 작성할 수 있다. 때로는 platform을 만들어주는데서 안만들기도 한다. 저수준에 해당되는 OpenMax IL을 사용해서도 media player를 만들 수 있기때문이다.

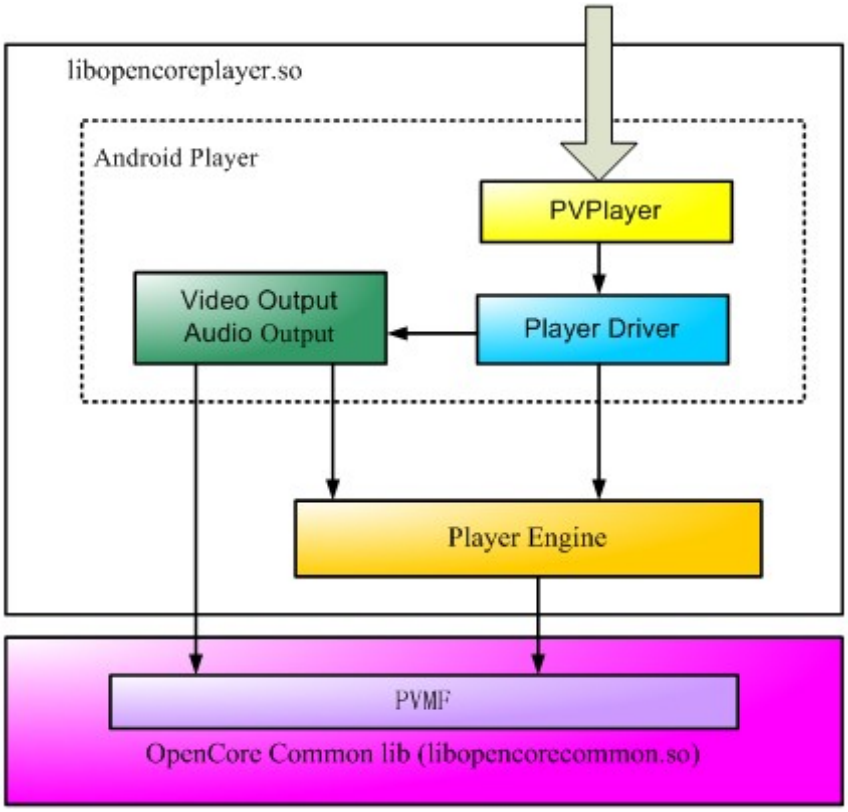
OpenMax IL : Integration Layer로써, multimedia codec들과 사용자들간의 interface를 제공한다. Component based program으로 부품을 설계하듯이 미디어 플레이어를 설계할 수 있다.

OpenMax DL : Development Layer로써, audio code,video codec들을 어떻게 만드는지 설명하고 있으며, 보통은 platform provider가 wrapping해서 OpenMax IL로 커버해주고 있기때문에, 미디어 플레이어 프로그래머가 볼필요는 거의 없다. (다만 어떻게 Codec등을 만드는지 관심있는 개발자는 참고로 할 수 있을 것 같다.).

5. OpenCore Player

5.1 Player 구성

OpenCore의 Player Makefile은 pvplayer/Android.mk이고, 컴파일이 끝나게 되면 libopencoreplayer.so 라이브러리 파일이 생성된다. 이 라이브러리는 두 개의 구성요소로 나눌 수가 있는데, 하나는 Player의 엔진에 대한 것이며, 다른 하나는 Android component player에 대한 것이다. 이것은 사실 adapter 개념이다.
player engine에 대한 경로는 external/opencore/engine/player 이고, adater에 대한 소스 경로는 external/opencore/adapter이다.



5.2 Player Engine부분

Player engine 부분은 아주 깔끔한 인터페이스를 갖는다. 위 그림에서 보듯이 다른 플레이어에 따른 다른 상황에 대처할 수 있도록 구성되어 있다.

engine 디렉토리의 구성은 다음과 같다.

```
engines/player/
|-- Android.mk
|-- build
|    |-- linux_nj
|    |-- make
|    `-- makefile.conf
|-- config
|    `-- linux_nj
|-- include
|    |-- pv_player_datasink.h
|    |-- pv_player_datasinkfilename.h
|    |-- pv_player_datasinkpvmfnode.h
|    |-- pv_player_datasource.h
|    |-- pv_player_datasourcepvmfnode.h
```

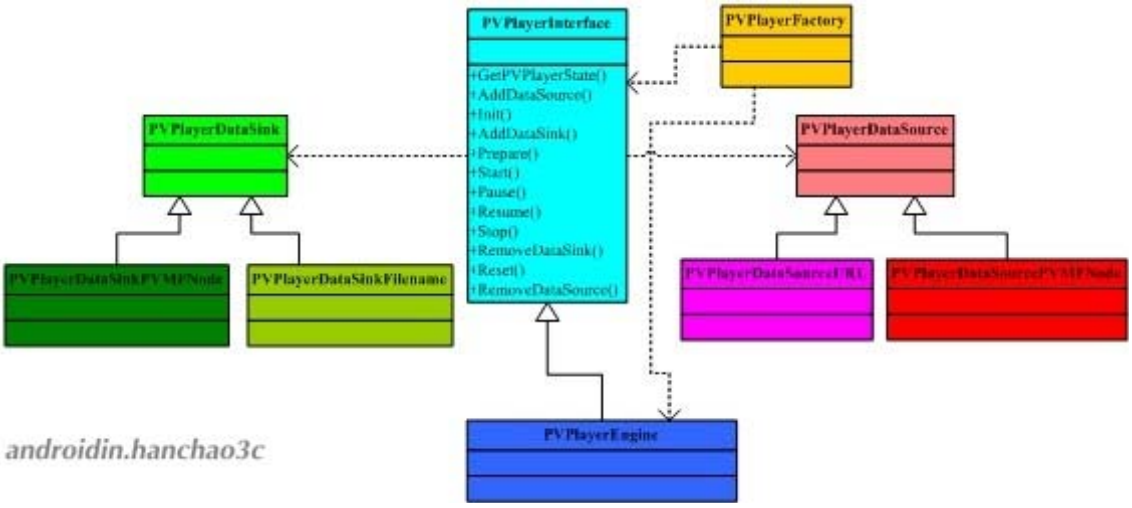
```
|   |-- pv_player_datasourceurl.h
|   |-- pv_player_events.h
|   |-- pv_player_factory.h
|   |-- pv_player_interface.h
|   |-- pv_player_license_acquisition_interface.h
|   |-- pv_player_registry_interface.h
|   |-- pv_player_track_selection_interface.h
|   `-- pv_player_types.h
|-- sample_app
|   |-- Android.mk
|   |-- build
|   |-- sample_player_app_release.txt
|   `-- src
|-- src
|   |-- pv_player_datapath.cpp
|   |-- pv_player_datapath.h
|   |-- pv_player_engine.cpp
|   |-- pv_player_engine.h
|   |-- pv_player_factory.cpp
|   |-- pv_player_node_registry.h
|   `-- pv_player_sdkinfo.h
`-- test
    |-- Android.mk
    |-- build
    |-- config
    `-- src
```

이 중 engines/player/include 디렉토리에는 인터페이스 관련 header파일들이 들어 있고, engines/player/src 디렉토리에는 소스들과 private header파일들이 있다.
여기서 중요한 헤더파일들은 다음과 같다.

pv_player_types.h: data structure들과 enumeration value 들에 대한 정의
pv_player_events.h: UUID 값과 각 에러값에 대한 정의
pv_player_datasink.h: datasink는 media data에 대한 출력을 얘기한다. PVPlayerDataSink에 대한 정의가 있고, 이것은 media data output 인터페이스에 대한 기본 class이다.
pv_player_datasinkfilename.h: the definition of succession PVPlayerDataSinkFilename category PVPlayerDataSink.
pv_player_datasinkpvmfnode.h: the definition of succession PVPlayerDataSinkPVMFNode category PVPlayerDataSink.
pv_player_datasource.h: datasource is the media data input, the definition of

category PVPlayerDataSource, this is the media data base class as interface.
pv_player_datasourcepvmfnode.h: the definition of succession PVPlayerDataSourcePVMFNode category PVPlayerDataSource.
pv_player_datasourceurl.h: the definition of succession PVPlayerDataSourceURL category PVPlayerDataSource.
pv_player_interface.h: the definition of Player interface PVPlayerInterface, this is an interface type.
pv_player_factory.h: the main factory class definition PVPlayerFactory, used to create and destroy PVPlayerInterface.

사실 engines/player/src 디렉토리의 메인은 PVPlayerEngine class가 정의되어 있는 pv_player_engine.cpp 이다.
PVPlayerEngine은 PVPlayerInterface로부터 상속된 클래스이고, 이것은 PVPlayerFactory가 생성이 될 때 PVPlayerInterface로 생성이 되고, 실제 생성은 PVPlayerEngine의 형태로 생성이 된다(pv_player_factory.cpp 파일을 보면 자세한 것을 알 수 있다)



Player 엔진은 구현되는 동안 code과 데이터 흐름제어 함수들, 그리고 미디어의 출력을 위해 외부에서 적절할게 세팅이 필요한 함수들이 설정이 된다. PVPlayerInterface 인터페이스는 이러한 것들을 위한 기본 동작에 대한 정의가 되어 있고, 그 메인인터페이스는 다음과 같다.

CODE:
PVCommandId AddDataSource(PVPlayerDataSource& aDataSource, const OsclAny* aContextData = NULL);
PVCommandId Init(const OsclAny* aContextData = NULL);

PVCommandId AddDataSink(PVPlayerDataSink& aDataSink, const OsclAny* aContextData = NULL);
PVCommandId Prepare(const OsclAny* aContextData = NULL);
PVCommandId Start(const OsclAny* aContextData = NULL);
PVCommandId Pause(const OsclAny* aContextData = NULL);
PVCommandId Resume(const OsclAny* aContextData = NULL);
PVCommandId Stop(const OsclAny* aContextData = NULL);
PVCommandId RemoveDataSink(PVPlayerDataSink& aDataSink, const OsclAny* aContextData = NULL);
PVCommandId Reset(const OsclAny* aContextData = NULL);
PVCommandId RemoveDataSource(PVPlayerDataSource& aDataSource, const OsclAny* aContextData = NULL);

여기에는 Video와 Audio 출력을 위하여 DataSink 관련 함수들을 포함하고 있다.
pv_player_types.h 에는 PVP_STATE_ 로 시작되는 player의 state machine의 상태를 나타내는 상태들이 선언되어 있다:

CODE:
```
typedef enum
{
    PVP_STATE_IDLE          = 1,
    PVP_STATE_INITIALIZED = 2,
    PVP_STATE_PREPARED     = 3,
    PVP_STATE_STARTED      = 4,
    PVP_STATE_PAUSED       = 5,
    PVP_STATE_ERROR        = 6
} PVPlayerState;
```

PVPlayerInterface는 각 동작이 성공하였을 경우 Player의 state machine을 변화시킬 수 있다.

player가 초기상태에 있을때 state는 PVP_STATE_IDLE이 되고,
Init 호출이 끝난 후에는 PVP_STATE_INITIALIZED 상태로 들어간다.
AddDataSink를 호출하는 것은 PVP_STATE_PREPARED로 접근하는 것이고,
Prepare를 호출한 후에는 PVP_STATE_PREPARED 상태로 들어간다.
start 함수를 호출해서 PVP_STATE_STARTED로 들어간 후
pause를 호출해서 PVP_STATE_PAUSED 상태로 될 수 있다.

PVP_STATE_STARTED, PVP_STATE_PAUSED 이 두 개의 상태는 play 상황에서 나오는 상태이며 각각 start와 pause 함수를 호출해서 이 두 개의 상태를 스위치 할 수 있다.
playback 상황에서는 stop을 호출해서 PVP_STATE_INITIALIZED 를 return할 수 있으며,

RemoveDataSource 함수를 호출해서 PVP_STATE_IDLE 상태를 return 할 수 있다.

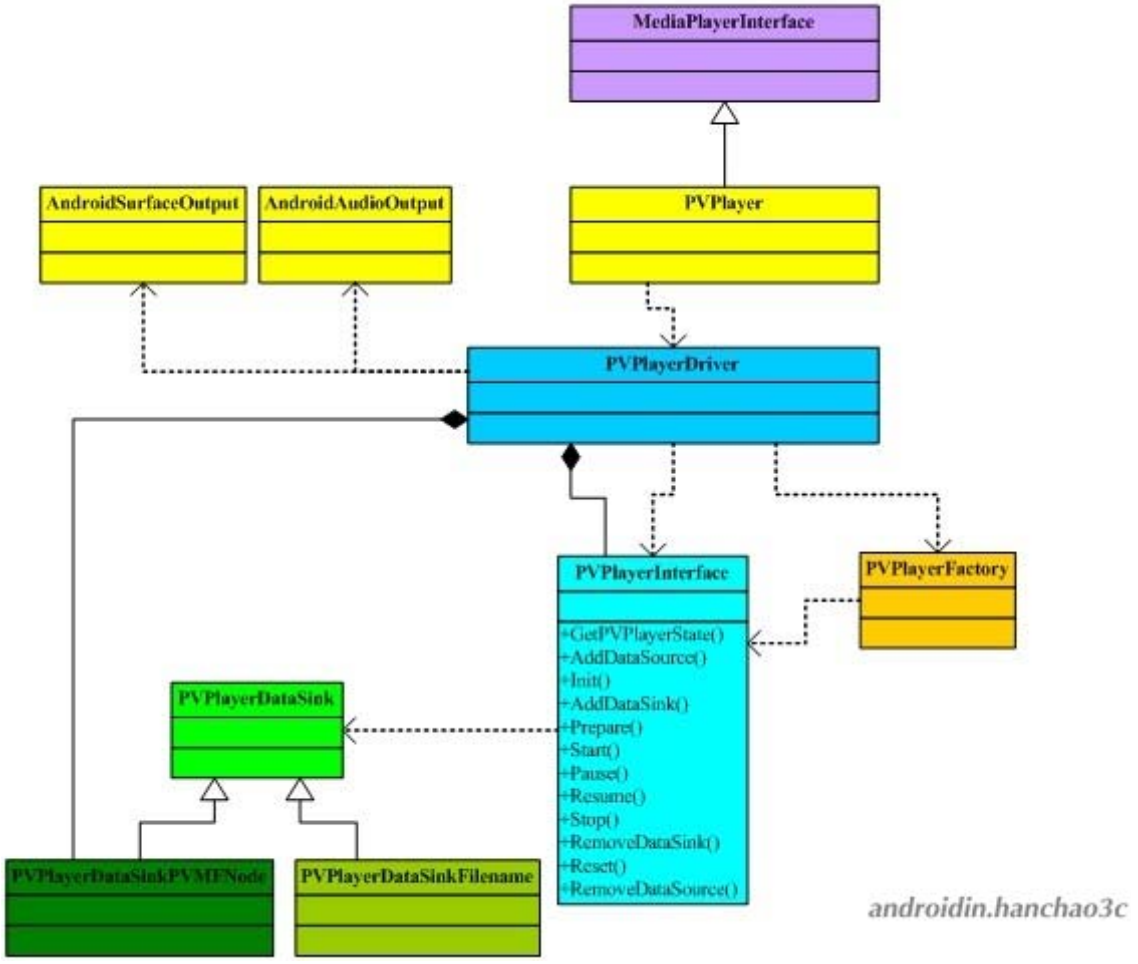
(참고로, pv_player_engine.cpp 내부적으로는 PVP_ENGINE_STATE_ 의 prefix의 state를 사용하고, 위의 PVP_STATE_ 의 경우는 PVP_ENGINE_STATE_ 의 상태를 적절히 체킹하여 state를 표시하게 되어 있다)

5.3 Android Player Adapter

Android 소스 중 안드로이드용 opencore player의 adapter 는 external/opencore/android 디렉토리에 있고, 그 파일 리스트는 다음과 같다.

```
android
|-- Android.mk
|-- android_audio_mio.cpp
|-- android_audio_mio.h
|-- android_audio_output.cpp
|-- android_audio_output.h
|-- android_audio_output_threadsafe_callbacks.cpp
|-- android_audio_output_threadsafe_callbacks.h
|-- android_audio_stream.cpp
|-- android_audio_stream.h
|-- android_log_appender.h
|-- android_surface_output.cpp
|-- android_surface_output.h
|-- mediascanner.cpp
|-- metadatadriver.cpp
|-- metadatadriver.h
|-- playerdriver.cpp
|-- playerdriver.h
`-- thread_init.cpp
```

Android의 Player의 adapter는 OpenCore player engine의 접속 interface를 호출해야한다. Android media player interface에 대한 구현은 PVPlayer의 구현이다. 사실 PVPlayer는 MediaPlayerInterface로부터 상속받은 것이다.
player 구현과정의 처음은 PlayerDriver를 얻어내는 것이고, 그리고 PVPlayer를 사용한다. PVPlayer는 PlayerDriver의 특정함수를 호출함으로써 생성되며, 전체 구현에 대한 구조는 다음과 같다.



PVPlayerDriver the various operations of the various commands used to complete these orders in the definition of playerdriver.h.

MediaPlayer에서 Player를 제어하기 위하여 PVPlayerDriver에는 다양한 동작에 대한 다양한 명령을 사용한다. 이런것은 playerdriver.h에 정의 되어 있다.

CODE:

```
// Commands that MediaPlayer sends to the PlayerDriver
enum player_command_type {
    PLAYER_QUIT                 = 1,
    PLAYER_SETUP                = 2,
    PLAYER_SET_DATA_SOURCE      = 3,
    PLAYER_SET_VIDEO_SURFACE    = 4,
    PLAYER_SET_AUDIO_SINK       = 5,
    PLAYER_INIT                 = 6,
```

```
    PLAYER_PREPARE                  = 7,
    PLAYER_START                    = 8,
    PLAYER_STOP                     = 9,
    PLAYER_PAUSE                    = 10,
    PLAYER_RESET                    = 11,
    PLAYER_SET_LOOP                 = 12,
    PLAYER_SEEK                     = 13,
    PLAYER_GET_POSITION             = 14,
    PLAYER_GET_DURATION             = 15,
    PLAYER_GET_STATUS               = 16,
    PLAYER_REMOVE_DATA_SOURCE       = 17,
    PLAYER_CANCEL_ALL_COMMANDS      = 18,
};
```

이러한 명령의 일반적인 구현은 PVPlayerInterface 각 부분에 대해서 간단한 인터페이스로 구현된다.

예를 들어 player를 잠시 멈추는(pause) 동작에 대해서 전체 시스템에서의 구현의 흐름은 다음과 같다.

1. PVPlayer 중의 pause 함수 구현(playerdriver.cpp 소스에서의)

CODE:
```
status_t PVPlayer::pause()
{
    LOGV("pause");
    return mPlayerDriver->enqueueCommand(new PlayerPause(0,0));
}
```

여기서 mPlayerDriver는 PlayerDriver 형의 변수이고, PlayerPause class를 하나 생성하면서 command를 큐에 집어넣는 역할을 한다. 여기서 PlayerPause class는 playerdriver.h 파일에 선언되어 있다.

2. PlayerDriver는 간접적으로 enqueueCommand 함수를 이용해서 메시지 처리에 대한 호출을 하고, PlayPause 명령에 대해서 결과적으로는 handlePause()함수를 호출하게 된다

CODE:
```
void PlayerDriver::handlePause(PlayerPause* ec)
{
    LOGV("call pause");
    mPlayer->Pause(0);
```

```
    FinishSyncCommand(ec);
}
```

여기서 mPlayer는 PVPlayerInterface형의 포인터이며, 이 포인터는 OpenCore의 player engine 타입인 PVPlayerEngine을 호출하기 위해서 사용된다.

player의 adapter 구현에 있어 Android framework에서 해야할 중요한 일은 media output을 정의하는 것이다(Audio/Vidoe output을 포함하는).
OpenCore는 Player Engine의 형태로 이러한 것들을 변환할 필요가 있다.
여기서 두 개의 중요한 타입이 존재하게 되는데, AndroidSurfaceOutput 을 담당하는 android_surface_output.cpp와 AndroidAudioOutput을 담당하는 android_audio_output.cpp가 그것이다.

Video output을 세팅하기 위해서는 PlayerDriver에 세 개의 멤버가 사용된다
(playerdriver.cpp에 있음)

CODE:
```
    PVPlayerDataSink        *mVideoSink;
    PVMFNodeInterface        *mVideoNode;
    PvmiMIOControl          *mVideoOutputMIO;
```

여기서
mVideoSink는 PVPlayerDataSink 타입이고, 이것은 player engine 인터페이스이다.
mVideoNode는 PVMFNodeInterface 타입이고, 이것은 external/opencore/pvmi/pvmf/include/pvmf_node_interface.h 에 정의되어 있으며 PVMF에 접근하는 모든 NODE에 통일된 인터페이스를 제공하는 클래스이다.
PvmiMIOControl타입인 mVideoOutputMIO 는 external/opencore/pvmi/pvmf/include//pvmi_mio_control.h 파일에 정의되어 있으며, 이 클래스는 midia input-output 제어를 위한 추상화된 인터페이스를 제공한다.

1. PVPlayer에 있는 setVideoSurface는 Video output interface를 설정한다. 여기서 사용되는 parameter는 ISurface 형의 포인터이다.

CODE:
```
status_t PVPlayer::setVideoSurface(const sp<ISurface>& surface)
{
    LOGV("setVideoSurface(%p)", surface.get());
    mSurface = surface;
    return OK;
```

}

setVideoSurface 함수는 PVPlayer의 멤버인 mSurface를 세팅한다(입력은 ISurface type 인 surface 임).

실제 video output 인터페이스에 대한 세팅은 run_set_video_surface() 함수에서 이루어진다.

CODE:
```
void PVPlayer::run_set_video_surface(status_t s, void *cookie)
{
    LOGV("run_set_video_surface s=%d", s);
    if (s == NO_ERROR) {
        PVPlayer *p = (PVPlayer*)cookie;
        if (p->mSurface == NULL) {
            run_set_audio_output(s, cookie);
        } else {
            p->mPlayerDriver->enqueueCommand(new PlayerSetVideoSurface(p->mSurface, run_set_audio_output, cookie));
        }
    }
}
```

이 함수는 void PVPlayer::run_init(status_t s, void *cookie, bool cancelled)
에서 호출되고

PlayerSetVideoSurface함수의 호출은 결국은 handleSetVideoSurface() 함수를 호출하게 된다.

2. handleSetVideoSurface함수의 구현은 다음과 같다.

CODE:
```
void PlayerDriver::handleSetVideoSurface(PlayerSetVideoSurface* ec)
{
    int error = 0;
    mVideoOutputMIO = new AndroidSurfaceOutput(ec->surface());
    mVideoNode = PVMediaOutputNodeFactory::CreateMediaOutputNode(mVideoOutputMIO);
    mVideoSink = new PVPlayerDataSinkPVMFNode;
```

```
    ((PVPlayerDataSinkPVMFNode *)mVideoSink)->SetDataSinkNode(mVideoNode);
    ((PVPlayerDataSinkPVMFNode *)mVideoSink)->SetDataSinkFormatType(PVMF_YUV420);

    OSCL_TRY(error, mPlayer->AddDataSink(*mVideoSink, ec));
    OSCL_FIRST_CATCH_ANY(error, commandFailed(ec));
}
// 참고: 현재(20090813)의 코드에서는 위의 코드가 많이 틀려졌다. 하지만 기본은 동일하다.
```

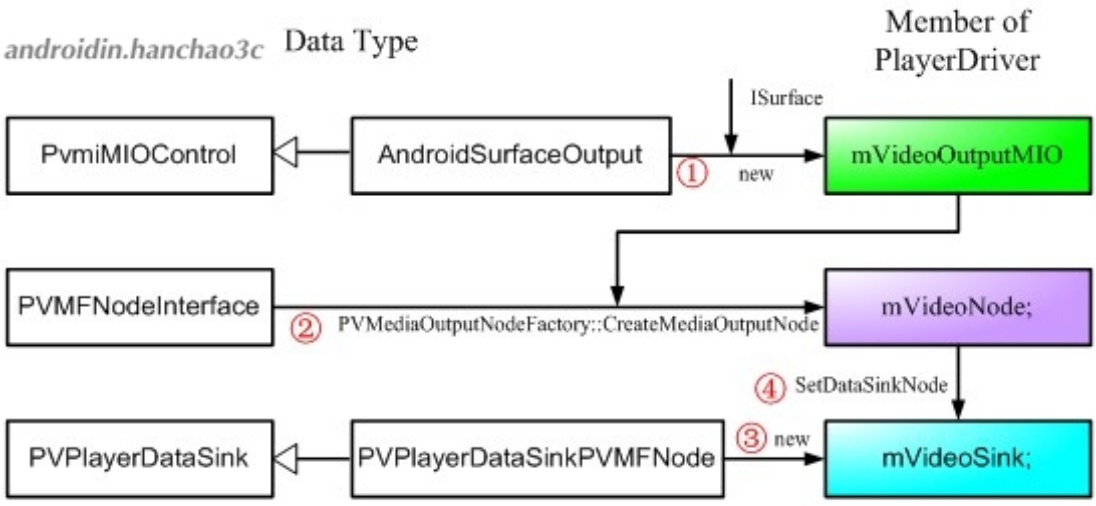
맨처음 여기서는 mVideoOutputMio(PvmiMIOControl type인)에 대한 설정이 이루어진다. 이것은 PvmiMIOConrol type으로부터 상속받은 AndroidSurfaceOut class를 가지고 이루어진다.
그리고, PVMediaOutputNodeFactory:: CreateMediaOutputNode() 함수를 호출함으로써 PVMFNodeInterface type인 mVideoNode와 연결을 하게 된다(입력은 mVideoOutputMIO가 된다)

그리고 난 후

PVPlayerDataSinkPVMFNode 형의 mVideoSink를 생성한다.
(./engines/player/include/pv_player_datasinkpvmfnode.h 에 선언되어 있음)
여기서 PVPlayerDataSinkPVMFNode class는 PVPlayerDataSink class로부터 상속된다.

그러므로, 이 mVideoSink는 PVPlayerDataSink로 사용될 수 있고, SetDataSinkNode 함수를 이용해서 mVideoNode를 mVideoSink의 output NODE로 세팅할 수 있다.

이것에 대한 class의 흐름은 다음과 같이 그릴 수 있다.



사실, Video Output에 대한 기본적인 함수들과 가능은 AndroidSurfaceOutput class에서 이루어진다. Player engine에서 Android에 적용하는 주요 부분은 ISurface 출력이다.

마지막에 있는 AddDataSink에 대한 호출은 mVideoSink을 PVPlayerInterface에 추가하는 것이다.
AndroidSurfaceOutput class에 대한 코드는 android_surface_output.cpp에 있다. 이 class에 대한 코드는 OpenCore Player engine의 video 출력과 Android “adapter"의 함수들과 동일한 구성이다(매칭이 된다).

AndroidSurfaceOutput class는 PvmiMIOControl class로부터 상속받은 상태이고, 이 class의 constructor의 파라미터는 ISurface type을 사용한다.

이 class(PvmiMIOControl)에 대한 구현은 모두 ISurface 인터페이스를 사용하는 것이 된다.

6. OpenCore author

media recording을 담당하는 android author 소스의 구성은 다음과 같다.

```
android/author/
|-- Android.mk
|-- android_audio_input.cpp
|-- android_audio_input.h
|-- android_audio_input_threadsafe_callbacks.cpp
|-- android_audio_input_threadsafe_callbacks.h
|-- android_camera_input.cpp
|-- android_camera_input.h
|-- authordriver.cpp
|-- authordriver.h
`-- mediarecorder.cpp
```

```
engines/author/
|-- Android.mk
|-- build
| |-- make
| `-- makefile
|-- include
| |-- pvauthorenginefactory.h
```

```
|  `-- pvauthorengineinterface.h
|-- src
|  |-- pvae_tuneables.h
|  |-- pvaenodeutility.cpp
|  |-- pvaenodeutility.h
|  |-- pvauthorengine.cpp
|  |-- pvauthorengine.h
|  `-- single_core
`-- test
|-- Android.mk
|-- build
|-- config
|-- src
`-- test_input
```