Standard for RealTek DVD Recordable

Date: 7/13/2011

DVD-VENUS Navigation Filter

RealTek specification on DVD Recordable Technology

Specification for DVD-VENUS: Navigation Filter

Note

This document is subject to review.

Chen Ma
chen.ma@realcomtec.com

Version	Date	Author
0.0.1	5/19/04	Chen Ma
0.0.2	1/6/05	Gordon Lai
0.0.3	5/26/05	Gordon Lai
0.0.4	6/22/05	Gordon Lai

Change List	Version	Description
	0.0.2	- Rename DVD Navigator to more general term “Navigation Filter”. - Document Input/Demux plug-in architecture for Navigation Filter - Document INavControl interface to replace older version
	0.0.3	- Add binary playlist structure for file input plugin
	0.0.4	- Add command line interface descriptions
		-

Table of Contents

1 List of Figures

2 Introduction

2.1 Purpose

The *Navigation Filter* is the generic source filter for playback flow of multiple media types. It opens all the necessary files, accepts user commands to navigate through the media and handle special presentation modes, reads and parses stream from media, and splits the stream into multiple elementary streams (such as audio, video, sub-picture) for the decoders.

2.2 Scope

This document covers the architecture and interface definition of Navigation Filter.

2.3 Glossary

- Flow: A collection of Filters and Pins that can perform a media-streaming task.
- OSAL: Operating System Abstraction Layer.

2.4 References

- Operating System Abstraction Layer (OSAL)
- Stream Class

2.5 Overview of Document

3 Generic Playback Flow and the Navigation Filter

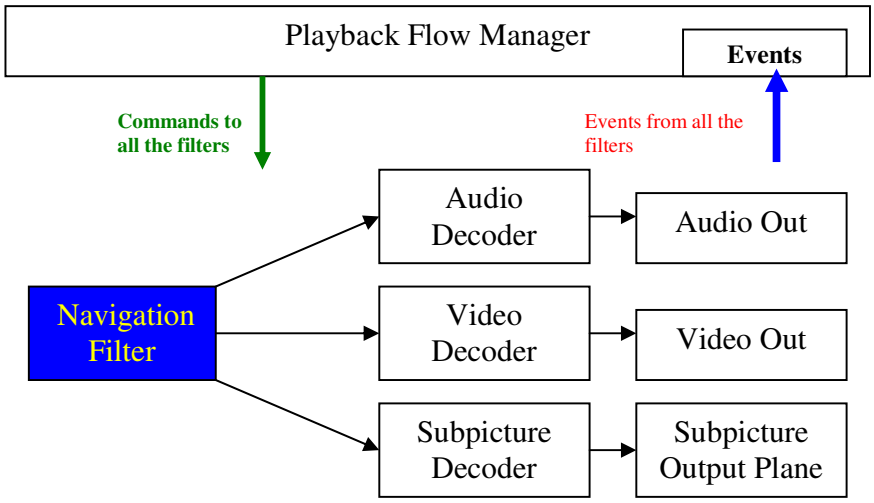


Figure 1 An Example of Playback Flow

Figure 1 demonstrates an example playback flow under StreamClass architecture. The Flow Manager is a class that serves as a gateway between Filters and Application. The Navigation filter is the source filter of the flow. It opens all the necessary files, accepts user commands to navigate through the media and handle special presentation modes, reads and parses stream from media, and splits the stream into multiple elementary streams (such as audio, video, sub-picture) for the decoders.

Creation of a Playback Flow

Application first creates all the instance of the filters. In here, they are Navigation Filter as the single source filter, and Audio/Video/Sub-picture decoding filters to handle decoding and feature rendering.

The Navigation Filter would provide control interface *INavControl* for application to send user navigation commands. Among these commands, *INavControl::LoadMedia* should be called first to allow Navigation Filter processing input/demux plug-in initialization according to the format of media.

Application then passes the reference of the filters to Flow Manager. Flow Manager behaves like a container; it collects the references to all the filters. The application then manually connects the filters through Flow Manager connection utility interfaces. Filter connects to other filters using Pin. When application sends a **Pause** command to the flow manager, the flow manager will send the command to all the filters under its collection. Application can receive the message via the Flow Manager event interface.

4 INavControl Interface for Navigation Commmands

CNavigationFilter derives from CBaseFilter, and also exposes another interface named INavControl. The class declaration is like,

class CNavigationFilter : public CBaseFilter, public INavControl

INavControl interface provides several methods that are categorized into the following chapters based on their functionalities.

4.1 Methods for Media Preparation and Configuration Setting

Method	Description
LoadMedia	Initialize the Navigation Filter according to the specified path to playback target
UnloadMedia	Un-initialize the Navigation Filter and release the resources
SetNavState	Restore the Navigation Filter internal state to a previously saved snapshot.
SetConfiguration	Configure the Navigation Filter to change its default behavior

4.1.1 INavControl::LoadMedia

Syntax

```
HRESULT LoadMedia(
        char* path);
```

Parameters

path

[in] the file system path of the playback target. For a disc type media, specify the folder path (such as VIDEO_TS for DVD-Video); For a file type media, specify the file path.

Return Value

Returns S_OK if successful, or an **HRESULT** value indicating the cause of the error.

4.1.2 INavControl::UnloadMedia

Syntax

```
HRESULT UnloadMedia();
```

Return Value

Returns S_OK if successful, or an **HRESULT** value indicating the cause of the error.

4.1.3 INavControl::SetNavState

Syntax

```
HRESULT SetNavState(
        void* state,
        unsigned int size);
```

Parameters

state

[in] point to the opaque navigator state structure.

size

[in] size of the opaque navigator state structure.

Return Value

Returns S_OK if successful, or an **HRESULT** value indicating the cause of the error.

4.1.4 INavControl::SetConfiguration

Syntax

```
HRESULT SetConfiguration(
        unsigned int configID,
        void* configData,
        unsigned int size);
```

Parameters

configID

[in] the unique configuration ID to identify the specific behavior to configure.

configData

[in] point to the data structure which defines the detail of configuration.

size

[in] size of configData.

Return Value

Returns S_OK if successful, or an **HRESULT** value indicating the cause of the error.

4.2 Methods for Playing Entities of Media Content

Method	Description
PlayTitle	Play from the title with specified index number.
PlayChapter	Play from the chapter with specified index number.
PlayNextChapter	Play from the next chapter related to the

	currently presented chapter
PlayPrevChapter	Play from the previous chapter related to the currently presented chapter
PlayAtTime	Play from the specified time point of a title
PlaySegment	Play a specified segment of a title (ex: from time point A to time point B)

4.2.1 INavControl::PlayTitle

Syntax

```
HRESULT PlayTitle(
        int titleNum,
        unsigned int cmdID);
```

Parameters

titleNum
[in] title number (-1: replay current title, or 1~N).
cmdID
[in] command identifier for caller to track command completion status.

Return Value

Returns S_OK if successful, or an **HRESULT** value indicating the cause of the error.

4.2.2 INavControl::PlayChapter

Syntax

```
HRESULT PlayChapter(
        int titleNum,
        int chapterNum,
        unsigned int cmdID);
```

Parameters

titleNum
[in] title number (-1: current title, or 1~N).
chapterNum
[in] chapter number (-1: replay current chapter, or 1~N).
cmdID
[in] command identifier for caller to track command completion status.

Return Value

Returns S_OK if successful, or an **HRESULT** value indicating the cause of the error.

4.2.3 INavControl::PlayNextChapter

Syntax

```
HRESULT PlayNextChapter(
```

```
        unsigned int cmdID);
```

Parameters
cmdID
[in] command identifier for caller to track command completion status.

Return Value
Returns S_OK if successful, or an **HRESULT** value indicating the cause of the error.

4.2.4 INavControl::PlayPrevChapter

Syntax
```
HRESULT PlayPrevChapter(
        unsigned int cmdID);
```

Parameters
cmdID
[in] command identifier for caller to track command completion status.

Return Value
Returns S_OK if successful, or an **HRESULT** value indicating the cause of the error.

4.2.5 INavControl::PlayAtTime

Syntax
```
HRESULT PlayAtTime(
        int titleNum,
        unsigned int eltmSeconds,
        unsigned int frameIdx,
        unsigned int cmdID;
```

Parameters
titleNum
[in] title number (-1: replay current title, or 1~N).
eltmSeconds
[in] elapsed play time in seconds.
frameIdx
[in] the frame index (within a second) to seek to (0~N).
cmdID
[in] command identifier for caller to track command completion status.

Return Value
Returns S_OK if successful, or an **HRESULT** value indicating the cause of the error.

4.2.6 INavControl::PlaySegment

Syntax
```
HRESULT PlaySegment(
        int titleNum,
```

```
        unsigned int startSeconds,
        unsigned int startFrameIdx,
        unsigned int endSeconds,
        unsigned int endFrameIdx,
        unsigned int cmdID);
```

Parameters
titleNum
[in] title number (-1: replay current title, or 1~N).
startSeconds, startFrameIdx
[in] start play position of segment.
endSeconds, endFrameIdx
[in] end play position of segment.
cmdID
[in] command identifier for caller to track command completion status.

Return Value
Returns S_OK if successful, or an **HRESULT** value indicating the cause of the error.

4.3 Methods for Menu Operation

Method	Description
MenuShow	Show menu specified by an ID number.
MenuEscape	Escape from the menu domain and re-enter previous domain.
ButtonSelectNumeric	Select a menu button with its index number.
ButtonSelectPoint	Select a menu button with (x, y) coordinates of a pointing device
ButtonActivateNumeric	Activate a menu button with its index number
ButtonActivatePoint	Activate a menu button with (x, y) coordinates of a pointing device
ButtonActivate	Activate the currently selected button
ButtonMoveSelection	Move button selection in the specified direction (up, down, left or right)

4.3.1 INavControl::MenuShow

Syntax
```
HRESULT MenuShow(
```

```
        unsigned int menuID,
        unsigned int cmdID);
```

Parameters
menuID
[in] menu identifier, which is defined by and can be queried from navigators.
cmdID
[in] command identifier for caller to track command completion status.

Return Value
Returns S_OK if successful, or an **HRESULT** value indicating the cause of the error.

4.3.2 INavControl::MenuEscape

Syntax
```
HRESULT MenuEscape(
        unsigned int cmdID);
```

Parameters
cmdID
[in] command identifier for caller to track command completion status.

Return Value
Returns S_OK if successful, or an **HRESULT** value indicating the cause of the error.

4.3.3 INavControl::ButtonSelectNumeric

Syntax
```
HRESULT ButtonSelectNumeric(
        int buttonNum,
        unsigned int cmdID);
```

Parameters
buttonNum
[in] button number (1~N).
cmdID
[in] command identifier for caller to track command completion status.

Return Value
Returns S_OK if successful, or an **HRESULT** value indicating the cause of the error.

4.3.4 INavControl::ButtonSelectPoint

Syntax
```
HRESULT ButtonSelectPoint(
        unsigned int x,
        unsigned int y,
        unsigned int cmdID);
```

Parameters
x, y
[in] point coordination used to identify a button.
cmdID
[in] command identifier for caller to track command completion status.

Return Value
Returns S_OK if successful, or an **HRESULT** value indicating the cause of the error.

4.3.5 INavControl::ButtonActivateNumeric

Syntax
```
HRESULT ButtonActivateNumeric(
        int buttonNum,
        unsigned int cmdID);
```

Parameters
buttonNum
[in] button number (1~N).
cmdID
[in] command identifier for caller to track command completion status.

Return Value
Returns S_OK if successful, or an **HRESULT** value indicating the cause of the error.

4.3.6 INavControl::ButtonActivatePoint

Syntax
```
HRESULT ButtonActivatePoint(
        unsigned int x,
        unsigned int y,
        unsigned int cmdID);
```

Parameters
x, y
[in] point coordination used to identify a button.
cmdID
[in] command identifier for caller to track command completion status.

Return Value
Returns S_OK if successful, or an **HRESULT** value indicating the cause of the error.

4.3.7 INavControl::ButtonActivate

Syntax
```
HRESULT ButtonActivate(
        unsigned int cmdID);
```

Parameters

cmdID
[in] command identifier for caller to track command completion status.

Return Value
Returns S_OK if successful, or an **HRESULT** value indicating the cause of the error.

4.3.8 INavControl::ButtonMoveSelection

Syntax
```
HRESULT ButtonMoveSelection(
        NAV_BTNDIR_ID direction,
        unsigned int cmdID);
```

Parameters
direction
[in] direction of button selection movement. See also NAV_BTNDIR_ID definition.
cmdID
[in] command identifier for caller to track command completion status.

Return Value
Returns S_OK if successful, or an **HRESULT** value indicating the cause of the error.

4.4 Methods for Special Navigation Operations

Method	Description
GoUp	Go to the upper playback unit of the current one.
StillEscape	Some media can freeze playback intentionally in a spontaneous STILL state. This is to resume playback from STILL state if escape is permitted.

4.4.1 INavControl::GoUp

Syntax
```
HRESULT GoUp(
        unsigned int cmdID);
```

Parameters
cmdID
[in] command identifier for caller to track command completion status.

Return Value
Returns S_OK if successful, or an **HRESULT** value indicating the cause of the error.

4.4.2 INavControl::StillEscape

Syntax

```
HRESULT StillEscape(
        unsigned int cmdID);
```

Parameters

cmdID

[in] command identifier for caller to track command completion status.

Return Value

Returns S_OK if successful, or an **HRESULT** value indicating the cause of the error.

4.5 Methods for Media Presentation Mode Change

Method	Description
SetVideoPresentationMode	Change video presentation mode.
SetAudioStream	Select presented audio stream.
SetAudioDownmixMode	Set Audio Down-mix mode.
SetSubpictureStream	Select presented sub-picture stream.
SetAngle	Select presented video angle.

4.5.1 INavControl::SetVideoPresentationMode

Syntax

```
HRESULT SetVideoPresentationMode(
        unsigned int mode,
        unsigned int cmdID);
```

Parameters

mode

[in] to be defined.

cmdID

[in] command identifier for caller to track command completion status.

Return Value

Returns S_OK if successful, or an **HRESULT** value indicating the cause of the error.

4.5.2 INavControl::SetAudioStream

Syntax

```
HRESULT SetAudioStream(
        int streamNum,
```

```
        unsigned int cmdID);
```

Parameters
streamNum
[in] stream number.
cmdID
[in] command identifier for caller to track command completion status.

Return Value
Returns S_OK if successful, or an **HRESULT** value indicating the cause of the error.

4.5.3 INavControl::SetAudioDownmixMode

Syntax
```
HRESULT SetAudioDownmixMode(
        unsigned int mode,
        unsigned int cmdID);
```

Parameters
mode
[in] to be defined.
cmdID
[in] command identifier for caller to track command completion status.

Return Value
Returns S_OK if successful, or an **HRESULT** value indicating the cause of the error.

4.5.4 INavControl::SetSubpictureStream

Syntax
```
HRESULT SetSubpictureStream(
        int streamNum,
        NAV_DISPLAY_STATE displayState,
        unsigned int cmdID);
```

Parameters
streamNum
[in] stream number.
displayState
[in] indicates display state as ON, OFF or unchanged. See also NAV_DISPLAY_STATE definition.
cmdID
[in] command identifier for caller to track command completion status.

Return Value
Returns S_OK if successful, or an **HRESULT** value indicating the cause of the error.

4.5.5 INavControl::SetAngle

Syntax

```
HRESULT SetAngle(
        int angleNum,
        unsigned int cmdID);
```

Parameters

angleNum
[in] angle number (1~N).
cmdID
[in] command identifier for caller to track command completion status.

Return Value

Returns S_OK if successful, or an **HRESULT** value indicating the cause of the error.

4.6 Methods for Information Enquiry

4.7 Operations via Interfaces Other than INavControl

4.8 Command Synchronization

5 Input and Demux Plug-in Architecture Inside Navigation Filter

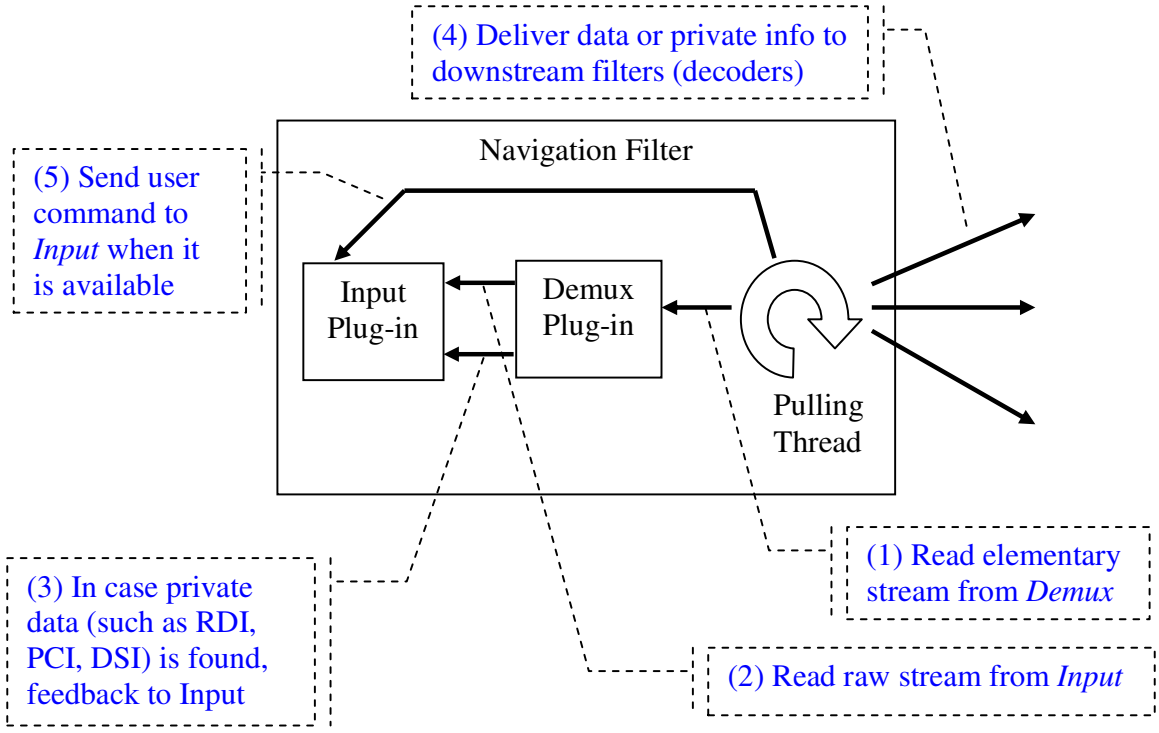


Figure 2 Input and Demux Plug-in Architecture

The Navigation Filter hosts Input and Demux plug-in internally. The selection of plug-ins (based on the format of media content), creation/initialization/disposal and configuration setting of plug-ins, are all done by the Navigation Filter.

The Navigation Filter will also maintain a *Pulling Thread* whenever it is not in STOP state. All the streaming and command processing activities should be done within the context of this thread. As a result, the Input and Demux plug-in are both working in passive mode, i.e., waiting for the Pulling Thread to call the functions it provides.

As depicted above, there are 5 major interactions that would happen between Navigation Filter and the Plug-ins:

(1) The Pulling Thread would constantly call Demux Plug-in's "read()" function whenever it likes to request elementary stream for playback.

(2) Mostly, Demux Plug-in then has to call Input Plug-in's "read()" function in order to request some "raw" stream from the target media. The only exception is when Demux Plug-in has some raw stream on site and not yet consumed.

(3) In some cases the Demux Plug-in can encounter some private data packets which is for Input Plug-in's information only (such as DVD-Video's PCI/DSI packets, DVD-VR's RDI packets). In that case, feedback the information to Input Plug-in.

(4) When the Pulling Thread returns from Demux Plug-in's "read()" function with valid data or private-info, it would deliver the stuff to the designated Navigation Filter output pin (which is connected).

(5) Meanwhile, application may send some user commands (via INavControl interface) to interrupt the playback. Navigation Filter will relay the commands to the Input Plug-in by calling its "execUserCmds()" function. Please note that the user command interrupts and routine data reading are from the same thread (the Pulling Thread), so they are mutual exclusive by the nature.

5.1 Input Plug-in Interface Structure

An Input Plug-in object and its functionality interface are exposed via the INPUTPLUGIN structure, which has the following members.

Member	Description
pInstance	A void* pointer represents object instance.
loadMedia	Ask input plug-in to initialize itself for the specified target media.
unloadMedia	Ask input plug-in to un-initialize itself and release to resources.
getDemuxType	Return DEMUX_PLUGIN_ID to request the suitable demux type.
read	Read data or commands from input plug-in.
updatePlaybackPosition	Update current playback position for input plug-in's reference.
privateDataFeedback	For some private data in stream which is for input plug-in's information (such as DSI, PCI, RDI), send back to input plug-in via this function.
execUserCmd	Execute User Commands to change the navigator behavior.
propertySetGet	Property set/get for input plug-in. Can be used to check/modify internal parameters, or request information about the media.

dispose Free the input plug-in object.

5.1.1 loadMedia

Syntax

```
HRESULT (*loadMedia)(
            void* pInstance,
            char* path);
```

Parameters

pInstance
[in] always pass the pInstance pointer from INPUTPLUGIN.
path
[in] the file system path of the playback target. For a disc type media, specify the folder path (such as VIDEO_TS for DVD-Video); For a file type media, specify the file path.

Return Value

Returns S_OK if successful, or an **HRESULT** value indicating the cause of the error.

5.1.2 unloadMedia

Syntax

```
HRESULT (*unloadMedia)(
            void* pInstance);
```

Parameters

pInstance
[in] always pass the pInstance pointer from INPUTPLUGIN.

Return Value

Returns S_OK if successful, or an **HRESULT** value indicating the cause of the error.

5.1.3 getDemuxType

Syntax

```
DEMUX_PLUGIN_ID (*getDemuxType)(
            void* pInstance);
```

Parameters

pInstance
[in] always pass the pInstance pointer from INPUTPLUGIN.

Return Value

Returns a DEMUX_PLUGIN_ID of the requested demux type.

5.1.4 read

Syntax

```
HRESULT (*read)(
            void* pInstance,
            NAVBUF* pBuffer);
```

Parameters
pInstance
[in] always pass the pInstance pointer from INPUTPLUGIN.
pBuffer
[out] a structure to receive data or commands read. See also NAVBUF definition.

Return Value
Returns S_OK if successful, or an **HRESULT** value indicating the cause of the error.

5.1.5 updatePlaybackPosition

Syntax
```
HRESULT (*updatePlaybackPosition)(
            void* pInstance,
            int64_t currentPTS);
```

Parameters
pInstance
[in] always pass the pInstance pointer from INPUTPLUGIN.
currentPTS
[in] current presentation-time-stamp used to indicate the current playback position.

Return Value
Returns S_OK if successful, or an **HRESULT** value indicating the cause of the error.

5.1.6 privateDataFeedback

Syntax
```
HRESULT (*privateDataFeedback)(
            void* pInstance,
            unsigned int id,
            unsigned char* data,
            unsigned int size);
```

Parameters
pInstance
[in] always pass the pInstance pointer from INPUTPLUGIN.
id
[in] identifier for private data type.
data
[in] data pointer to the beginning of private data.
size
[in] number of bytes pointed by data.

Return Value

Returns S_OK if successful, or an **HRESULT** value indicating the cause of the error.

5.1.7 execUserCmd

Syntax

```
HRESULT (*execUserCmd)(
            void* pInstance,
            NAV_CMD_ID id,
            void* cmdData,
            unsigned int cmdDataSize,
            unsigned int* pIsFlushRequired);
```

Parameters

pInstance
[in] always pass the pInstance pointer from INPUTPLUGIN.
id
[in] identifier of command type. See also NAV_CMD_ID definition.
cmdData
[in] data pointer to the command arguments.
cmdDataSize
[in] size of command arguments totally in bytes.
pIsFlushRequired
[out] to respond whether a FLUSH operation is required immediately.

Return Value

Returns S_OK if successful, or an **HRESULT** value indicating the cause of the error.

5.1.8 propertySetGet

Syntax

```
HRESULT (*propertySetGet)(
            void* pInstance,
            NAV_PROP_ID id,
            void* propertyData,
            unsigned int propertyDataSize);
```

Parameters

pInstance
[in] always pass the pInstance pointer from INPUTPLUGIN.
id
[in] identifier for the property type. See also NAV_PROP_ID definition.
propertyData
[in/out] data pointer to the property value.
propertyDataSize
[in] size of property value.

Return Value

Returns S_OK if successful, or an **HRESULT** value indicating the cause of the error.

5.1.9 dispose

Syntax

```
HRESULT (*dispose)(
            void* pInstance);
```

Parameters

pInstance

[in] always pass the pInstance pointer from INPUTPLUGIN.

Return Value

Returns S_OK if successful, or an **HRESULT** value indicating the cause of the error.

5.2 Demux Plug-in Interface

A demux Plug-in object and its functionality interface are exposed via the DEMUXPLUGIN structure, which has the following members.

Member	Description
pInstance	A void* pointer represents object instance.
getOutputs	Get number of output sources from demux, and the corresponding descriptive names.
read	Read data or commands from demux plug-in.
flush	Navigation Filter will use this function to reset buffered raw stream inside demux plug-in.
propertySetGet	Property set/get for demux plug-in. Can be used to check/modify internal parameters, or request information about stream.
dispose	Free the demux plug-in object.

5.2.1 getOutputs

Syntax

```
unsigned int (*getOutputs)(
            void* pInstance,
            const char*** pppOutputNames);
```

Parameters

pInstance

[in] always pass the pInstance pointer from DEMUXPLUGIN.

pppOutputNames
[in] return an array of text strings for descriptive names of each outputs.

Return Value
Returns number of output sources from the demux plug-in.

5.2.2 read

Syntax
```
HRESULT (*read)(
            void* pInstance,
            NAVBUF* pBuffer);
```

Parameters
pInstance
[in] always pass the pInstance pointer from DEMUXPLUGIN.
pBuffer
[out] a structure to receive data or commands read. See also NAVBUF definition.

Return Value
Returns S_OK if successful, or an **HRESULT** value indicating the cause of the error.

5.2.3 flush

Syntax
```
HRESULT (*flush)(
            void* pInstance);
```

Parameters
pInstance
[in] always pass the pInstance pointer from DEMUXPLUGIN.

Return Value
Returns S_OK if successful, or an **HRESULT** value indicating the cause of the error.

5.2.4 propertySetGet

Syntax
```
HRESULT (*propertySetGet)(
            void* pInstance,
            NAV_PROP_ID id,
            void* propertyData,
            unsigned int propertyDataSize);
```

Parameters
pInstance
[in] always pass the pInstance pointer from DEMUXPLUGIN.
id
[in] identifier for the property type. See also NAV_PROP_ID definition.

propertyData
[in/out] data pointer to the property value.
propertyDataSize
[in] size of property value.

Return Value
Returns S_OK if successful, or an **HRESULT** value indicating the cause of the error.

5.2.5 dispose

Syntax

```
HRESULT (*dispose)(
            void* pInstance);
```

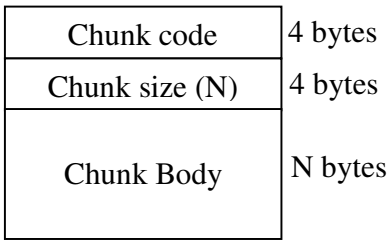
Parameters
pInstance
[in] always pass the pInstance pointer from DEMUXPLUGIN.

Return Value
Returns S_OK if successful, or an **HRESULT** value indicating the cause of the error.

6 Binary Playlist Structure for File Input Plug-in

In addition to basic file playback supported by file input plug-in, some advanced editing features require more "accurate" descriptions on how the specific media source should be handled, as well as some proprietary commands for the underlying decoder/renderer.

It's defined herein a flexible binary playlist structure to contain detailed descriptions and proprietary commands. The general building block of the playlist structure is called a "chunk", which has the following structure,



Chunk code consists of 4 characters (FourCC code) and is used to identify the type of chunk.

Chunk size immediately follows *Chunk code*, it specifies the number of bytes in *Chunk Body*.

Chunk Body contains informative binary structures. The format of this binary structure depends on the *Chunk code*.

6.1 Playlist Chunk Definitions

The following Chunk codes are already defined,

Chunk Code	*Chunk Body Format Description*
' file '	A null terminated string of the path to the file source
' info '	The file playback session information, the structure is illustrated below as NAVFILESESSIONINFO (defined in "NavDef.h")
' init '	Specify the decode-init command for pass-through to decoder / renderer. (defined in "PrivateInfo.h, as VIDEODECODEINIT)
' dcod '	The decode command for pass-through to decoder / renderer. (defined in "PrivateInfo.h, as VIDEODECODECOMMAND)
' blnd '	The blend command for transition effect or seamless playback. (defined in "PrivateInfo.h, as VIDEOBLENDCOMMAND)

The NAVFILESESSIONINFO is defined as,

```
typedef struct _tagNAVFILESESSIONINFO {

    int64_t startAddress;
    int64_t endAddress;
    int64_t stcOffset;
    int channelIndex;
    ENUM_MEDIA_TYPE mediaType;

} NAVFILESESSIONINFO;
```

"*startAddress*" and "*endAddress*" are the beginning and ending addresses (relative to head of the file source) of the playback session ; *"stcOffset"* specifies stc_offset for this file playback session, this value will be added to original bit-stream PTS for actual PTS; "*channelIndex"* specifies the assigned channel for this file playback session ; "*mediaType*" is specified if the format of file source is already known, otherwise, it should be set to "*MEDIATYPE_None*".

6.2 Playlist Chunk Ordering

Only "file" chunk and "info" chunk are mandatory. For the optional chunks (decode/blend/init commands), there can be from 0 to multiple chunks for one playlist entry.

A playlist entry should always begin with a "file" chunk, and an "info" chunk follows immediately. After these two mandatory chunks, the other optional chunks can be presented in any order.

Here is an example of the playlist entries

Entry #1					Entry #2		Entry #3			
file	info	init	Dcod	blnd	file	info	file	info	dcod	blnd

6.3 Playlist loading

The INavControl interface provides a function called "LoadMedia", which is normally used to load a single media. However, it can also be used in the playlist case as described above.

Here we proposed adding a "prefix" before the path to playlist binary file. File input plug-in can use this prefix to distinguish it from normal media files. For example,

pNavControl->LoadMedia(“playlist://./mytestingplaylist”);

can be used to load a playlist binary file. The binary file can reside on memory mapped file system for better efficiency.

7 Navigation Filter Command Line Interface

```
# Command Line Commands

   help    - Show the list of commands
             (operand) none

   load    - Load media from a path (INavControl::LoadMedia)
             (operand) @path : the absolute path to where the media is stored
             (example) load /dev/dvd

   unload  - Release the currently loaded media (INavControl::UnloadMedia)
             (operand) none

   setget  - The generic property set/get function (INavControl::SetGetProperty)
             (operand) @id       : a number to identify which property to set/get
                       @property : a number to assign to the property (for set only)
             (example) set 0 1
             (NOTE***) specific property id and meaning of values are NOT defined yet

   getps   - Print out the current playback status (INavControl::GetPlaybackStatus)
             (operand) none
             (example) a sample print out

                domain             = 1    # 0:STOP, 1:TITLE, 2:MENU, 3:VR_PG, 4:VR_PL
                numTitles          = 36   # number of titles in the whole media
                currentTitle       = 25   # current title index (1 ~ number of titles)
                numChapters        = 1    # number of chapters in the current title
                currentChapter     = 1    # current chapter index (1 ~ number of chapters)
                numAngles          = 9    # number of angles in the current title
                currentAngle       = 1    # current angle index (1 ~ number of angles)
                elapsedTime        = 1/15 # current elapsed time (see NOTE*** below)
                totalTime          = 80/0 # total time of current title (see NOTE*** below)
                currChapterStartTime = 0/0  # start time of current chapter (see NOTE*** below)
                currChapterEndTime   = 80/0 # end time of current chapter (see NOTE*** below)
                bPaused            = 0    # 1:playback is PAUSED, 0:otherwise
                bInAngleBlock      = 0    # 1:in multi-angle block, 0:otherwise
                speed              = 256  # current speed setting (256 = 1X)
                skip               = 0    # current skip setting (definition to be clarified)
                repeatMode         = 0    # 0:none, 1:Title-Repeat, 2:Chapter-Repeat, 3:AB-Repeat
                mediaType          = 34   # media-type number (an ENUM value defined elsewhere)
                languageCode       = 'en' # preferred menu language code (ISO-639 2-letter-lowercase symbols)

             (NOTE***) the TIME format (m/n) consists of two values
                    m: total number of seconds (0~), for example, 3600 is for a hour
                    n: the frame index within a second. For NTSC, it can be 0~29, and for PAL it's 0~24

   getvs   - Print out the video stream(s) status (INavControl::GetVideoStatus)
             (operand) none
             (example) a sample print out

                numStreams        = 1 # total number of video streams (for now, it should be always 1)
                indexCurrentStream = 1 # current stream index (1 ~ number of streams)
                presentationMode   = 0 # video presentation mode (0:NORMAL, 1:WIDE, 2:PAN-SCAN, 3:LETTER-BOX)
                stream #1
                   type            = 29     # video media-type number (an ENUM value defined elsewhere)
                   tvSystem        = 0      # TV system (0:NTSC, 1:PAL)
                   aspectRatio     = 4/3    # aspect ratio (width/height)
                   frameSize       = 720/480 # video frame size (width/height)
                   srcLtrBox       = 0      # whether it came with original letter box (0:NONE, 1~7:otherwise)
                   line21Switch1   = 0      # 1:line-21 user data recorded for field 1, 0:otherwise
                   line21Switch2   = 0      # 1:line-21 user data recorded for field 2, 0:otherwise
                   bAllowPanScanMode  = 0      # 1:PAN-SCAN presentation mode should be allowed, 0:otherwise
                   bAllowLetterBoxMode = 0     # 1:LETTER-BOX presentation mode should be allowed, 0:otherwise

   getas   - Print out the audio stream(s) status (INavControl::GetAudioStatus)
             (operand) none
             (example) a sample print out
```

```
    numStreams        = 8 # total number of audio streams (1~8)
    indexCurrentStream = 1 # current stream index (1 ~ number of streams)
    stream #1
      type         = 17    # audio media-type number (an ENUM value defined elsewhere)
      languageCode  = 'jp'  # audio stream language (ISO-639 2-letter-lowercase symbols)
      description   = 0     # audio stream description (see NOTE*** below)
      bitsPerSample = 16    # number of bits per sample (0:unknown)
      samplingRate  = 48000 # sampling rate (0:unknown)
      numChannels   = 2     # number of audio channels (0:unknown)
    stream #2
      ...

  (NOTE***) audio stream description codes

    0: not specified
    1: normal audio
    2: audio for visually impaired
    3: director's comments

getss   - Print out the sub-picture stream(s) status (INavControl::GetSubpictureStatus)
  (operand) none
  (example) a sample print out

    numStreams        = 32 # total number of sub-picture streams (1~32)
    indexCurrentStream = 0  # current stream index (1 ~ number of streams)
    bDummyStream      = 0  # 1:Dummy stream is selected, 0:otherwise
    bDisplay          = 0  # 1:Sub-picture display is ON, 0:OFF
    stream #1
      type         = 22   # sub-picture media-type number (an ENUM value defined elsewhere)
      languageCode  = 'en' # sub-picture stream language (ISO-639 2-letter-lowercase symbols)
      description   = 0    # sub-picture stream description (see NOTE*** below)
    stream #2
      ...

  (NOTE***) sub-picture stream description codes

    0:  not specified
    1:  normal sub-picture
    2:  bigger sub-picture
    3:  sub-picture for children
    4:  normal closed-captioning sub-picture
    5:  bigger closed-captioning sub-picture
    6:  closed-captioning sub-picture for children
    7:  forced sub-picture
    8:  normal director's comments sub-picture
    9:  bigger director's comments sub-picture
    10: director's comments sub-picture for children

getms   - Print out status of available menus (INavControl::GetMenuStatus)
  (operand) none
  (example) a sample print out

    numMenus = 6   # total number of available menus
      menuID = 2 # available menu ID (see NOTE*** below)
      menuID = 3 # available menu ID (see NOTE*** below)
      menuID = 4 # available menu ID (see NOTE*** below)
      menuID = 5 # available menu ID (see NOTE*** below)
      menuID = 6 # available menu ID (see NOTE*** below)
      menuID = 7 # available menu ID (see NOTE*** below)

  (NOTE***) meaning of defined DVD-Video menu IDs

    2: title menu
    3: root menu
    4: sub-picture menu
    5: audio menu
    6: angle menu
    7: chapter menu

getds   - Print out the current disc (media) status (INavControl::GetDiscStatus)
```

(operand) none
(example) a sample print out

allowedRegions = FFh # bit-mask to indicate playback is allowed for which regions (see NOTE*** below)
discType = 23 # main media-type number (an ENUM value defined elsewhere)
discSubtype = 34 # sub media-type number (an ENUM value defined elsewhere)

(NOTE***) bit 0 (LSB) shows the status of region 1, bit 1 for region 2, and so force.
DVD-Video spec divided th world into 8 regions.
For example, allowedRegions 05h (0000 0101b) means disc playable in region 1 (US) and 3 (Taiwan)

getts - Print out the status of a specific title (INavControl::GetTitleStatus)
(operand) @titleNum : index number of the specific title of interest
(example) getts 3
a sample print out

numChapters = 1 # number of chapters in the specific title
numAngles = 1 # number of angles in the specific title
totalTime = 60/0 # total time of the specific title (see NOTE*** of "getps" command above)

getcs - Print out the status of the latest executed command (INavControl::GetLatestCmdStatus)
(operand) none
(example) a sample print out

executedCmdType = 13 # type ID of the latest executed command (defined elsewhere)
executedCmdID = 8 # ID (serial numbering) of the latest executed command
executedCmdResult = SUCCESS (10000000h) # result of the latest executed command
numOfPendingCmds = 0 # number of commands waiting in the queue (not yet executed)

getns - Get binary block that represents current navigator state (INavControl::GetNavState)
(NOTE***) NOT SUPPORTED YET !!!

setns - Assign a saved binary block for navigator to restore its previous state (INavControl::SetNavState)
(NOTE***) NOT SUPPORTED YET !!!

playt - Play from the start of a specific title (INavControl::PlayTitle)
(operand) @titleNum : title index number (-1: the current title)
(example) playt 3 # play from the start of title 3
playt -1 # play from the start of the current title
(NOTE***) if current domain is not TITLE (such as MENU or STOP), use -1 for the operands would result in
rejection of command.

playc - Play from the start of a specific chapter (INavControl::PlayChapter)
(operand) @titleNum : title index number (-1: the current title)
@chapterNum : chapter index number (-1: the current chapter)
(example) playc 3 8 # play from the start of title 3 chapter 8
playc -1 5 # play from the start of chapter 5 of current title
playc -1 -1 # play from the start of current chapter
playc 3 -1 # ILLEGAL FORMAT!
(NOTE***) if current domain is not TITLE (such as MENU or STOP), use -1 for the operands would result in
rejection of command.

next - Play from the start of next chapter (INavControl::PlayNextChapter)
(operand) none

prev - Play from the start of previous chapter (INavControl::PlayPrevChapter)
(operand) none

time - Play from a specific frame (specified by elapsed time) of a title (INavControl::PlayAtTime)
(operand) @titleNum : title index number (-1: the current title)
@startSeconds : elapsed time in seconds
@startFrameIdx : frame index within a second (0~29 for NTSC, 0~24 for PAL)
(example) time 3 1234 0 # play from the 1st frame since 20th minute 34th second, of title 3
time -1 3600 3 # play from the 4th frame since 1st hour, of the current title

seg - Play from a specific frame to another (both specified by elapsed time) of a title (INavControl::PlaySegment)
(NOTE***) NOT SUPPORTED YET !!!

menu - Jump into a menu (INavControl::MenuShow)
(operand) @menuID : the destination menu ID (see NOTE*** of "getms" command above)

menuesc - Go back to where we were before jumping into menu (TITLE or STOP) (INavControl::MenuEscape)
(operand) none

selnum - Select a menu button (highlight it) with numeric keys (INavControl::ButtonSelectNumeric)
(operand) @buttonNum : index number of the specific button
(NOTE***) NOT SUPPORTED YET !!!

selpt - Select a menu button (highlight it) with (x,y) coordinate (pointing device) (INavControl::ButtonSelectPoint)
(operand) @x : x coordinate
@y : y coordinate
(example) select 360 240 # select the point in the middle of a NTSC TV
(NOTE***) NOT SUPPORTED YET !!!

actnum - Activate a menu button (click it) with numeric keys (INavControl::ButtonActivateNumeric)
(operand) @buttonNum : index number of the specific button
(NOTE***) NOT SUPPORTED YET !!!

actpt - Activate a menu button (click it) with (x,y) coordinate (pointing device) (INavControl::ButtonActivatePoint)
(operand) @x : x coordinate
@y : y coordinate
(NOTE***) NOT SUPPORTED YET !!!

act - Activate the menu button (click it) which is currently selected (highlighted) (INavControl::ButtonActivate)
(operand) none

move - Change selected menu button by specifying direction of highlight movement
(INavControl::ButtonMoveSelection)
(operand) @direction : 0:UP, 1:DOWN, 2:RIGHT, 3:LEFT
(example) move 3 # move menu highlight to the left

return - Go back to the parent layer of menu (or title content) hierarchy (INavControl::GoUp)
(operand) none
(example) return # we may go back to root menu from chapter menu

stillesc - Break the STILL (where audio/video presentation froze intentionally) and keep on playing
(INavControl::StillEscape)
(operand) none

setvpm - Set video presentation mode (INavControl::SetVideoPresentationMode)
(operand) @mode : 0:NORMAL, 1:WIDE, 2:PAN-SCAN, 3:LETTER-BOX
(NOTE***) NOT SUPPORTED YET !!!

setas - Select active audio stream (INavControl::SetAudioStream)
(operand) @streamNum : audio stream number (1~8)
(example) setas 2 # switch to audio stream number 2

setad - Set audio downmixing mode (INavControl::SetAudioDownmixMode)
(operand) @mode : NOT DEFINED YET !!!
(NOTE***) NOT SUPPORTED YET !!!

setss - Select active sub-picture stream, or turn it on/off (INavControl::SetSubpictureStream)
(operand) @streamNum : sub-picture stream number (1~32)
@displayState : 0:OFF, 1:ON, 2:UNCHANGED
@bDummyStream : 1:use dummy stream, 0:otherwise
(example) setss 8 0 0 # switch to sub-picture stream number 8, keep display OFF
setss 3 1 0 # switch to sub-picture stream number 3, turn ON display
setss 7 2 0 # don't change display state, simply switch to sub-picture stream number 7
setss -1 2 1 # don't change display state, use "dummy" stream mode (the 1st operand "streamNum" is
ignored)

angle - Switch to a specific angle (INavControl::SetAngle)
(operand) @angleNum : target angle number (1~9)
(example) angle 6 # switch to angle 6
(NOTE***) the switch-angle behavior may be different depending on the multi-angle media characteristics

If the current position is NOT within a multi-angle block (please refer to "bInAngleBlock" in "getps" command)
Nothing happens
Else If the multi-angle stuff is created as "non-seamless"

Angle switches immediately, video may stutter, and it may jump backward a couple seconds in the new angle

Else If the multi-angle stuff is created as "seamless"

Angle may not switch immediately but in a couple seconds. Audio/video transition between angles should be smooth