

Further Player Software

API Reference

Issue: 01

Date: 2012.11.22

www.dokodemoterebi.com

About This Document

Purpose

This document describes the related product versions, intended audience and document contents.

Related Versions

The following table lists the product version related to this document.

| Product Name | Version |
|---|---------|
| Further Player Software Development Kit (SDK) | V1.0.0 |

Intended Audience

This document describes the reference information developed based Further Player, and it is intended for the programmers who meet the following requirements:

- Be familiar with C/C++ programming language
- Be familiar with 32-bit Windows environment/ MFC

Organization

The document first summarizes the Further Player API function types and their connections, and then describes the using method of each function in detail.

This document is organized as follows:

| Chapter | Content |
|-----------------------------|--|
| 1 Overview | Describes the SDK components and development environments of the Further Player. |
| 2 API Description | Provides an overview of Further Player SDK API, and describes all the APIs functions in detail. |
| 3 Enumerated Type | Describes the enumerated types used by Further Player. |
| 4 API Application Instances | Describes the examples of live streaming playing, local recorded files playback and real-time recording. |

1 Overview

This chapter describes the main components of Further Player API, including four components: playing live streaming, TimeShift playback, real-time recording (.mp4 format) and recorded files playback.

1.1 Player SDK components

| Component | Item | Description |
|----------------------|---|--|
| API interface | FG_AV_Manager.h | API interface should be included in the user project. |
| Static guide library | FurtherPlayer.lib | Static library should be linked in the user project. And the “dll” file and executive programs should be saved in the same folder. |
| Dynamic library | FurtherPlayer.dll | |
| Decoding Filters | FGAudioDecode.ax FGVideoDecode.ax MP4\$plitter.ax | Used when playing recorded files (mp4 format), and it has to register with system first. |
| Example code | Please refer to API application examples | Compiled by VS2005 |

1.2 Function list

1.2.1 Live streaming playing

| Function name | Description |
|----------------------------|--|
| SetPlayWnd | Sets the video playing window handle and main window handle. |
| ChangeVideoWindow | Adjusts video playing window ratio. The ratio value is: 0: 4/3; 1: 16:9; other: default |
| ReSizeWindow | Adjusts video ratio. The ratio value is: 0: 4/3; 1: 16:9; other: default |
| RePaintVideo | Refreshes the video window |
| GetVideoRect | Gets coordinate of the video playing window: x, y, cx, xy. |
| GetVideoAspectRatio | Gets the aspect ratio of the current video. |
| GetAVState | Returns with the AV running state |
| SetAVState | Sets the AV running state |
| CreateNetSource | Creates the network playing source |
| SetResolution | Sets the video resolution and TV system of network data streams. |
| ReBuildNetGraph | Recreates Graph after setting video resolution. |
| Video_Write_RingBuffer | Writes the network video data. |
| Audio_Write_RingBuffer | Writes the network audio data. |
| RB_GetMaxVideoWritePercent | Returns with the valid data size written by video RingBuffer. |
| RB_GetMaxAudioWriteSize | Returns with the valid data size written by audio RingBuffer. |
| ClearVideoBuffer | Clears video RingBuffer data. |
| ClearAudioBuffer | Clears audio RingBuffer data. |
| SetSound | Sets the playing volume value. |
| SetMute | Mute or un-mute. |
| AVSnap | Screenshot |
| ChangeSource | Video source switch (CV/SV/TV). Destroys NetGraph and then recreates it. |
| ChangeChannel | Channel switch, mainly used to clear the internal AV RingBuffer data and Dec RingBuffer. |

1.2.2 TimeShift playing

| Function name | Description |
|-------------------|---|
| OpenTimeShift | Starts to use TimeShift or stops using TimeShift. |
| GetTSBufSize | Gets the size of TimeShift buffer file. |
| SetTSPlayPos | Sets the TimeShift playback position. |
| GetTSPlayPos | Gets the TimeShift playback position. |
| TimeShiftPlay | Plays the TimeShift file. |
| TimeShiftPasue | Pauses playing the TimeShift file. |
| SetTimeShiftState | Enables or disables the TimeShift function. |

1.2.3 Real-time recording

| Function name | Description |
|----------------|----------------------------------|
| AVRecord | Starts or stops local recording. |
| SetRecordPause | Pauses recording. |

1.2.4 Recorded files playback

| Function name | Description |
|------------------------|-------------------------------------|
| CreateFileSource | Creates the file playback source. |
| FileGetCurrentPosition | Gets the current playback duration. |
| FileSetCurrentPosition | Sets the current playback duration. |
| FileGetDuration | Gets the file's total duration. |
| FileStop | Stops playing. |
| FilePlay | Starts to play. |
| FilePause | Pause playing. |

1.2.5 Callback function

| Function name | Description |
|---------------|---|
| CallBackFunc | Local module callback function, the address of callback function will be written by user. |

2 API Description

2.1 Live streaming playing functions

2.1.1 HRESULT SetPlayWnd(HWND **const** &inPlayWnd,HWND **const** &inMainWnd);

| | |
|---------------|---|
| Purpose: | Sets the video playing window handle and main window handle. |
| Parameter: | Input: inPlayWnd: Video playing window handle inMainWnd: Mainwindow handle Output: NULL |
| Return Value: | S_OK: Successful; not S_OK: Failed |

2.1.2 HRESULT ChangeVideoWindow(**int const** inVideoRatio,HWND **const** &inPlayWnd);

| | |
|---------------|---|
| Purpose: | Adjusts video playing window ratio. The ratio value is: 0: 4/3; 1: 16:9; other: default |
| Parameter: | Input: inVideoRatio, 0: 4/3; 1: 16:9 inPlayWnd: Video playing window handle Output: NULL |
| Return Value: | S_OK: Successful; not S_OK: Failed |

2.1.3 HRESULT ReSizeWindow(**int const** inVideoRatio);

| | |
|---------------|--|
| Purpose: | Adjusts video ratio. The ratio value is: 0: 4/3; 1: 16:9; other: default |
| Parameter: | Input: inVideoRatio, 0: 4/3; 1: 16:9 Output: NULL |
| Return Value: | S_OK: Successful; not S_OK: Failed |

2.1.4 HRESULT RePaintVideo(HWND **const** inVideoHWND);

| | |
|---------------|---|
| Purpose: | Refreshes the video window |
| Parameter: | Input: inVideoHWND, Video playing window handle Output: NULL |
| Return Value: | S_OK: Successful; not S_OK: Failed |

2.1.5 HRESULT GetVideoRect(RECT &outRect);

| | |
|---------------|--|
| Purpose: | Gets coordinate of the video playing window: x, y, cx, xy. |
| Parameter: | Input: NULL |
| | Output: RECT(x,y,cx,xy) |
| Return Value: | S_OK: Successful; not S_OK: Failed |

2.1.6 AV_VideoAspectRatio GetVideoAspectRatio(void){return m_eVideoAspectRatio;};

| | |
|---------------|---|
| Purpose: | Gets the aspect ratio of the current video. |
| Parameter: | Input: NULL |
| | Output: NULL |
| Return Value: | Please refer to the AV_VideoAspectRatio type description. |

2.1.7 HRESULT GetAVState(AV_RunState &outAVState);

| | |
|---------------|---|
| Purpose: | Returns with the AV running state |
| Parameter: | Input: NULL |
| | Output: Please refer to the AV_RunState type description. |
| Return Value: | S_OK: Successful; not S_OK: Failed |

2.1.8 void SetAVState(AV_RunState &inAVState);

| | |
|---------------|---|
| Purpose: | Sets the AV running state |
| Parameter: | Input: NULL |
| | Output: Please refer to the AV_RunState type description. |
| Return Value: | NULL |

2.1.9 HRESULT CreateNetSource(void);

| | |
|---------------|------------------------------------|
| Purpose: | Creates the network playing source |
| Parameter: | Input: NULL |
| | Output: NULL |
| Return Value: | S_OK: Successful; not S_OK: Failed |

2.1.10 HRESULT SetResolution(Resolution_ratio const inResolution,int inPalNtsc);

| | |
|---------------|---|
| Purpose: | Sets the video resolution and TV system of network data streams. |
| Parameter: | Input: inResolution: resolution (please refer to Resolution_ratio) inPalNtsc: TV system, 0: NTSC; 1: PAL |
| | Output: NULL |
| Return Value: | S_OK: Successful; not S_OK: Failed |

2.1.11 HRESULT ReBuildNetGraph(void);

| | |
|---------------|---|
| Purpose: | Recreates Graph after setting video resolution. |
| Parameter: | Input: NULL |
| | Output: NULL |
| Return Value: | S_OK: Successful; not S_OK: Failed |

2.1.12 BOOL Video_Write_RingBuffer(unsigned char *inVideoData,unsigned int inlen,unsigned int *inRealWritelen);

| | |
|---------------|--|
| Purpose: | Writes the network video data. |
| Parameter: | Input: inVideoData: Video data; inlen: Video data length |
| | Output: inRealWritelen: Return with the length of written data |
| Return Value: | TRUE: Successful; FALSE: Failed |

2.1.13 BOOL Audio_Write_RingBuffer(unsigned char *inAudioData,unsigned int inlen);

| | |
|---------------|--|
| Purpose: | Writes the network audio data. |
| Parameter: | Input: inAudioData: Audio data; inlen: Audio data length |
| | Output: NULL |
| Return Value: | TRUE: Successful; FALSE: Failed |

2.1.14 unsigned int RB_GetMaxVideoWritePercent(void);

| | |
|---------------|--|
| Purpose: | Returns with the valid data size written by video RingBuffer |
| Parameter: | Input: NULL |
| | Output: NULL |
| Return Value: | 0-100 |

2.1.15 unsigned int RB_GetMaxAudioWriteSize(void);

| | |
|---------------|---|
| Purpose: | Returns with the valid data size written by audio RingBuffer. |
| Parameter: | Input: NULL |
| | Output: NULL |
| Return Value: | 0-100 |

2.1.16 HRESULT ClearVideoBuffer(void);

| | |
|---------------|------------------------------------|
| Purpose: | Clears video RingBuffer data. |
| Parameter: | Input: NULL |
| | Output: NULL |
| Return Value: | S_OK: Successful; not S_OK: Failed |

2.1.17 HRESULT ClearAudioBuffer(void);

| | |
|---------------|------------------------------------|
| Purpose: | Clears audio RingBuffer data. |
| Parameter: | Input: NULL |
| | Output: NULL |
| Return Value: | S_OK: Successful; not S_OK: Failed |

2.1.18 HRESULT SetSound(long const inSound, BOOL const isMute = FALSE);

| | |
|---------------|---|
| Purpose: | Sets the playing volume value. |
| Parameter: | Input: inSound: Volume value range: 0~100 isMute: 0: Un-mute; 1: Mute |
| | Output: NULL |
| Return Value: | S_OK: Successful; not S_OK: Failed |

2.1.19 HRESULT SetMute(BOOL isMute);

| | |
|---------------|---|
| Purpose: | Mute or un-mute. |
| Parameter: | Input: isMute: TRUE: Mute; FALSE: Un-mute |
| | Output: NULL |
| Return Value: | S_OK: Successful; not S_OK: Failed |

2.1.20 HRESULT AVSnap(WCHAR *inFileName);

| | |
|---------------|--|
| Purpose: | Screenshot |
| Parameter: | Input: inFileName: Screenshot file path and file name. Output: NULL |
| Return Value: | S_OK: Successful; not S_OK: Failed |

2.1.21 HRESULT ChangeSource(void);

| | |
|---------------|--|
| Purpose: | Video source switch (CV/SV/TV). Destroys NetGraph and then recreates it. |
| Parameter: | Input: NULL Output: NULL |
| Return Value: | S_OK: Successful; not S_OK: Failed |

2.1.22 HRESULT ChangeChannel(void);

| | |
|---------------|--|
| Purpose: | Channel switch, mainly used to clear the internal AV RingBuffer data and Dec RingBuffer. |
| Parameter: | Input: NULL Output: NULL |
| Return Value: | S_OK: Successful; not S_OK: Failed |

2.2 TimeShift playing functions

2.2.1 HRESULT OpenTimeShift(BOOL inIsOpen,AV_StreamType inStreamType=eNetTimeShift);

| | |
|---------------|---|
| Purpose: | Starts to use TimeShift or stops using TimeShift. |
| Parameter: | Input: inIsOpen:TRUE: Starts to use TimeShift; FALSE: Stops using TimeShift inStreamType: TimeShift type (Please refer to AV_StreamType) |
| | Output: NULL |
| Return Value: | S_OK: Successful; not S_OK: Failed |

2.2.2 HRESULT GetTSBufSize(unsigned int &outSize);

| | |
|---------------|---|
| Purpose: | Gets the size of TimeShift buffer file. |
| Parameter: | Input: NULL |
| | Output: outSize: Buffer file size |
| Return Value: | S_OK: Successful; not S_OK: Failed |

2.2.3 HRESULT SetTSPlayPos(unsigned int &inPos);

| | |
|---------------|--|
| Purpose: | Sets the TimeShift playback position. |
| Parameter: | Input: inPos: Sets the TimeShift playback position |
| | Output: NULL |
| Return Value: | S_OK: Successful; not S_OK: Failed |

2.2.4 HRESULT GetTSPlayPos(unsigned int &outPos);

| | |
|---------------|--|
| Purpose: | Gets the TimeShift playback position. |
| Parameter: | Input: NULL |
| | Output: outPos: Sets the TimeShift playback position |
| Return Value: | S_OK: Successful; not S_OK: Failed |

2.2.5 HRESULT TimeShiftPlay(void);

| | |
|---------------|--|
| Purpose: | Plays the TimeShift file, mainly used when it switches from “Pause” state to “Play” state. |
| Parameter: | Input: NULL Output: NULL |
| Return Value: | S_OK: Successful; not S_OK: Failed |

2.2.6 HRESULT TimeShiftPasue(void);

| | |
|---------------|---|
| Purpose: | Pauses playing the TimeShift file, mainly used when it switches from “Play” state to “Pause” state. |
| Parameter: | Input: NULL Output: NULL |
| Return Value: | S_OK: Successful; not S_OK: Failed |

2.2.7 HRESULT SetTimeShiftState(BOOL isOpen,AV_StreamType streamType = eNetTimeShift);

| | |
|---------------|---|
| Purpose: | Enables or disables the TimeShift function. |
| Parameter: | Input: isOpen = True: Enables TimeShift; isOpen= False: Disables TimeShift streamType: TimeShift live streaming or network streaming Output: NULL |
| Return Value: | S_OK: Successful; not S_OK: Failed |

2.3 Real-time recording functions

2.3.1 HRESULT AVRecord(BOOL isRecord, WCHAR* inRecordFileName = _T("C:\\Further.mp4"), BOOL const isCutOff = TRUE, unsigned int minutes = 120);

| | |
|---------------|---|
| Purpose: | Starts or stops local recording. |
| Parameter: | Input: isRecord True: Starts recording; False: Stops recording; inRecordFileName: Recorded files path isCutOff: TRUE: Cuts the recorded file at every preset duration, then continues recording and saves to another recorded file. isCutOff: False: The maximum recorded file size is 4G. |
| | Output: NULL |
| Return Value: | S_OK: Successful; not S_OK: Failed |

2.3.2 void SetRecordPause(BOOL isPause);

| | |
|---------------|---|
| Purpose: | Pauses recording. |
| Parameter: | Input: True: Pauses recording; False: Continues recording |
| | Output: NULL |
| Return Value: | S_OK: Successful; not S_OK: Failed |

2.4 Recorded files playback functions

2.4.1 HRESULT CreateFileSource(WCHAR* mFileName = _T("C:\\Further.mp4"));

| | |
|---------------|------------------------------------|
| Purpose: | Creates the file playback source. |
| Parameter: | Input: mFileName: The file path |
| | Output: NULL |
| Return Value: | S_OK: Successful; not S_OK: Failed |

2.4.2 HRESULT FileGetCurrentPosition(double &outPosition);

| | |
|---------------|--|
| Purpose: | Gets the current playback duration. |
| Parameter: | Input: NULL |
| | Output: outPosition: The unit is second. |
| Return Value: | S_OK: Successful; not S_OK: Failed |

2.4.3 HRESULT FileSetCurrentPosition(double inPosition);

| | |
|---------------|--|
| Purpose: | Sets the current playback duration. |
| Parameter: | Input: inPosition: The unit is second. |
| | Output: NULL |
| Return Value: | S_OK: Successful; not S_OK: Failed |

2.4.4 HRESULT FileGetDuration(double & outDuration);

| | |
|---------------|---|
| Purpose: | Gets the file's total duration. |
| Parameter: | Input: NULL |
| | Output: outDuration: The file's total duration, its unit is second. |
| Return Value: | S_OK: Successful; not S_OK: Failed |

2.4.5 HRESULT FileStop(void);

| | |
|---------------|------------------------------------|
| Purpose: | Stops playing. |
| Parameter: | Input: NULL |
| | Output: NULL |
| Return Value: | S_OK: Successful; not S_OK: Failed |

2.4.6 HRESULT FilePlay(void);

| | |
|---------------|------------------------------------|
| Purpose: | Starts to play. |
| Parameter: | Input: NULL |
| | Output: NULL |
| Return Value: | S_OK: Successful; not S_OK: Failed |

2.4.7 HRESULT FilePause(void);

| | |
|---------------|------------------------------------|
| Purpose: | Pause playing. |
| Parameter: | Input: NULL |
| | Output: NULL |
| Return Value: | S_OK: Successful; not S_OK: Failed |

2.5 Callback function

2.5.1 int CallBackFunc(PFCALLBACK Func);

| | |
|---------------|---|
| Purpose: | Local module callback function, the address of callback function will be written by user. |
| Parameter: | Input: The address of callback function |
| | Output: NULL |
| Return Value: | Return value: 1 |

3 Enumerated Type

3.1 AV_SourceType data streaming type

```
typedef enum
{
    eNoSource = 0,           // No source data
    eNetSource,              // Network source data
    eNetFileSource,          // Network file source
    eLocalFileSource,        // Local file source
    eTimeShiftSource,        // TimeShift source
}AV_SourceType;            // Defines the data streaming type of the source created by AV
```

3.2 AV_RunState playing state mark

```
typedef enum
{
    eAV_Unused = 0,          // Initial state
    eAV_Stop ,               // Stop state
    eAV_Pause,               // Pause state
    eAV_Play,                // playing state
    eAV_Fast,
    eAV_Slow,
    eAV_Record
}AV_RunState;              // Playing state mark
```

3.3 Resolution_ratio resolution mark

```
typedef enum
{
    QCIF_Resolution_ratio = 0 , // 176×64K~K
    CIF_Resolution_ratio ,      // 352×256K~K
    Half_Resolution_ratio ,     // 704×768K~.5M
    DCIF_Resolution_ratio ,     // 528×512K~M
    D1_Resolution_ratio         // 704×1M~M以上
}Resolution_ratio ;           // Resolution mark
```


3.4 Video rendering mode

```
typedef enum
{
    VMR7Renderer = 0,
    VMR9Renderer,
    EVRRenderer,           // Enhanced video renderer
    VideoRenderer,        // The traditional rendering mode
    NORenderer
}AV_RenderMode;
```

3.5 Detection system type

```
typedef enum
{
    win_xp=0,
    win_vista,
    win_7,
    win_server2003,
    win_server2000,
    win_unknown
}AV_OSType;
typedef void (WINAPI *PGNSI)(LPSYSTEM_INFO);
```

3.6 Network streaming type

```
typedef enum
{
    eNetStream = 0,           // Network live streaming
    eNetTimeShift,           // TimeShift
    eNetFileStream,          // Network file streaming
    eFileStream               // Local file
}AV_StreamType;
```

3.7 Video aspect ratio definition

```
typedef enum
{
    eAspectRatioDefault = 0,      // Default
    eAspectRatio43 ,              // 4:3
    eAspectRatio169                // 16:9
}AV_VideoAspectRatio;
```

3.8 TV system description

```
typedef enum
{
    AV_NTSC=0,                    // NTSC TV system
    AV_PAL                          // PAL TV system
}AV_SYS_PAL_NTSC;
```

4 API Application Instances

4.1 Playing live streaming

Live streaming playing has to be used together with 8960AX network module. Live streaming playing system only enables after 8960AX network module writes AV data, as the following example:

```
Main()
{
    FG_AV_Manager *AV_Manager = new FG_AV_Manager();
    AV_Manager->CallBackFunc(HandleAVCallback);    // UI callback function
    // Sets AV resolution
    AV_Manager->SetResolution(CIF_Resolution_ratio); // Sets according to the written video resolution
    AV_Manager->SetRenderMode(VMR7Renderer);    // Sets video rendering mode
    AV_Manager->SetPlayWnd(HWND videoHwnd,HWND mainHwnd);
    If(AV_Manager->CreateNetSource())
    {
        // User can create thread and write audio and video data.
        While(1)
        {
            // Writes audio and video data separately
            AV_Manager-> Video_Write_RingBuffer(unsigned char *inVideoData,unsigned int
inlen,unsigned int *inRealWritelen);
            AV_Manager-> Audio_Write_RingBuffer (unsigned char *inAudioData,unsigned int inlen);
        }
    }
    Delete AV_Manager;
}
```

4.2 TimeShift playing

```
Main()
{
    FG_AV_Manager *AV_Manager = new FG_AV_Manager();
    AV_Manager->CallBackFunc(HandleAVCallback);    // UI callback function
    // Sets AV resolution
    AV_Manager->SetResolution(CIF_Resolution_ratio); //Sets according to the written video resolution
    AV_Manager->SetRenderMode(VMR7Renderer);    // Sets video rendering mode
    AV_Manager->SetPlayWnd(HWND videoHwnd,HWND mainHwnd);
    // Sets TimeShift
    AV_Manager->SetTimeShiftState(TRUE);
    If(AV_Manager->CreateNetSource())
    {
        // User can create thread and write audio and video data.
        While(1)
        {
            // Writes audio and video data separately.
            AV_Manager-> Video_Write_RingBuffer(unsigned char *inVideoData,unsigned int
inlen,unsigned int *inRealWritelen);
            AV_Manager-> Audio_Write_RingBuffer (unsigned char *inAudioData,unsigned int inlen);
        }
    }
    Delete AV_Manager;
}
```

4.3 Real-time recording

The example of real-time recording when playing live streaming is as following:

Starts to record: AV_Manager->AVRecord(TRUE);

Stops recording: AV_Manager->AVRecord(FALSE);

4.4 Recorded files playback

Main()

```
{
    FG_AV_Manager *AV_Manager = new FG_AV_Manager();
    AV_Manager->CallBackFunc(HandleAVCallback);    // UI callback function
    AV_Manager->SetRenderMode(VMR7Renderer);    // Sets video rendering mode
    AV_Manager->SetPlayWnd(HWND videoHwnd,HWND mainHwnd);
    If(AV_Manager->CreateFileSource())
    {
        // Pause or play operation
    }
    Delete AV_Manager;
}
```

