VNDP VIVOTEK Network Development Platform

VIVOTEK NETWORK DEVELOPMENT PLATFORM

VNDP - MainProfile

2009/8/31

TABLE OF CONTENTS

1. Overview

1.1. Introduction

VNDP MainProfile is DLL based developing tools supported by VIVOTEK to his Application Development Partners (ADPs) to help them quickly develop application that can use the maximum power of VIVOTEK Visual Server series. In this document, we will explain the concept of MainProfile and the relationship among the various modules in MainProfile. For details of each module, please refer to the document of individual module.

1.2. Why Use MainProfile

MainProfile hides many complexities of interaction with VIVOTEK's Visual Server series. It tries to unify the programming interfaces used to talk with Visual Server, so that it is possible to minimize the efforts that ADPs needed to pay when writing their applications.

For the developers that need to focus on the knowledge domain of their business fields, MainProfile provides a very efficient default implementation of live video/audio collecting and presentation. The implementation tries to use the maximum hardware capabilities on the client systems so it could save you a lot of time to tune the performance.

2. Getting Started

2.1. MainProfile Architecture

Architecture Overview for MainProfile

MainProfile contains 7 different modules. They provide the following functionalities:

DataBroker

Handles the media data connection with Visual Server. It output the data from callback functions in frame mode. So that users don't have to handle connection handshaking and the parsing and reassembling of media frame. It also provides a tunnel for HTTP only media (video only) to use the ability for frame reassemble capability. In this case, users need to handle the network connection on their own.

AVSyncrhonizer

Handles the decoding for encoded frames (either live media stream from Visual Server or stored media in storage) and the synchronization of audio and video data. Since this module uses the DirectX as the underlying video output mechanism, it could take advantage of the hardware acceleration ability in the system hence achieve a great performance. And this module also provides another channel which bypass the output of media and give users the decoded media frames. In this case, users could use the decoded frames to do their own business.

ServerManager

Provides a wrapper for the access of some common used URLs. Such as DI/DO, Uart (COM port on servers) read/write, config.ini retrieving and updating(for 2K, 3K series), set parameters and get parameters (for 6K, 7K, 8K series) and PTZ camera control. And if these functions can't fit your requirements, this module also provides a function let you do HTTP POST or GET, so that you could send to server whatever command you need to talk to it.

DataPacketParser

Parses the raw frame data got from storage, and turns it into a readable structure. In this structure, it contains the useful information about the frame. Since the input interface of AVSynchronizer needs structure rather than raw frame data, it is application's responsibility to call parsing when doing media playback from database.

PacketMaker

Provides the capability to regenerate packet format from user's encoded data. This enables users to develop their own proxy server by converting MPEG4 to MJPEG packets or could generate client side audio packets so that the packet could be used in two ways communication. The latter one is not implemented yet. This module is only available for Windows 32 platform now.

DRMControl

Provides developers the way to search the servers that are currently running on LAN.

To use this module, you must ensure that the servers or IP cameras have equipped with DRM enabled firmware. This module is only available for Windows 32 platform now.

ServerChannel

Provides the capability of Event tunnel, Control tunnel and UART tunnel. With those tunnel mechanism, user can receive Event notification without the streaming. And user can send control command (such as PTZ, SetDO) via tunnel for faster response than the traditional HTTP POST/GET.

SrvDepResource (deprecated, please use ServerManager)

Provides the supported model list and the capabilities of each model. With this module, it is possible to write highly model independent application since the capabilities of server could be determined right after the connection established. With the capabilities information, you could choose the right flow sequence dynamically. This module will be updated whenever the product list changed. When this happens, what application needs to do is to update this DLL. With no or few changes the application will be able to handle with new server models.

ServerUtl (deprecated, please use ServerManager)

Provides a wrapper for the access of some common used URLs. Such as DI/DO, Uart (COM port on servers) read/write, config.ini retrieving and updating, and PTZ camera control. And if these functions can't fit your requirements, this module also provides a function let you do HTTP POST or GET, so that you could send to server whatever command you need to talk to it.

2.2. Sample Code

In the SDK package, we provide many samples to introduce you the usage of MainProfile, please refer to the package for the samples.

2.3. MainProfile Essentials

Handle-based operation

For some MainProfile modules (DataBroker, AVSynchronizer, and ServerManager) that need to store information between various function calls, we use handle to carry the information. That's why you need to create the handle and pass these handle value to latter functions. Handle is noting but a pointer to an internal structure, users don't need to know the details of this pointer, but it is users' responsibility to free these handles (by calling the Delete or Release functions in the modules) whenever you finish using them. If you don't free them, memory would be leaked.

Users should be aware that it is a serious mistake if you pass a handle created in one module to the function in another module. This would lead to application crash. Even in the same module, there are something more than one kind of handles. Don't mix them together.

Also have to be noticed that some free functions need the handle itself, but some other need the pointer to the handle. For the latter one, the free function will clear the content of the handle to 0, and the former will not.

The other 2 modules (DataPacketParser and SrvDepResource) don't use handle since the functions provided are not related to each other.

Version Information

When using the modules, it is very important to make sure to use the correct version of DLLs. In MainProfile we store the version information in two places: In the resource of the DLLs, and also in a function (in xx_GetVersionInfo format) that could retrieve the version number. Users need to call the retrieval function and compare the version number with the constant defined in the header file of the module that is using. For handle-based module, this work is done automatically when you call Initial function.

There are four fields in the version information. They are major, minor, build and revision. Change of major version number often means change of existing interface or changes for function parameters. So if detected with different major version number, the initial function will return error. Minor version number usually means add of functions. So for application compiled with lower minor version number, it is ok to use DLLs with higher minor version number as long as the major number is the same. But if the initial function finds that the minor version number application passed is higher than the number recorded in DLL, the initial function will be failed since the application might use functions that doesn't not implemented or defined in this DLL.

Build and revision numbers are for tracking the releases of the DLL. Higher revision always means the module have something improved, such as some bugs are fixed or the performance is improved.

For the two modules without handle in function interface, it is up to application to do the check.

Callback Based Flow

For DataBroker and AVSynchronizer, they will create internal threads to handle the individual works. So when they need to notify users about the status or send the media data to users, they use callback functions. Callback functions are nothing but a pointer to a function implemented by application and called by MainProfile modules.

Since the MainProfile defines the callback function with "__stdcall" calling convention, applications that implement the callback function must also declare the function with this calling convention. If not, the program will be crashed when the callback function got called. Note if you set the project setting to compile the whole project using "__stdcall" convention, it is not necessary to add __stdcall preposition when you implement the function.

The callbacks are not always called by a separate thread, some functions such as the decoder channel defined in AVSynchronizer, the callback is actually called using the thread users input frame. We use callback here is to unify the interface for different codec type. For motion JPG, the decoder out the decoded data right after the decoding finished. But for H263 and MPEG, the output will be one frame lagged. When the data are output by callback, application could ignore the difference since it always handle the output of the decoded data by in the callback function that is always asynchronous.

When handling with callback function, it is very important to notice that don't put time -consuming work in the callback function unless the callback is called by users' thread.

Online V.S. Offline Settings

In DataBroker, the unit for handling with one camera is a connection. If you want to get the live media data from the camera, you need to create a connection and then connect to server. The connection has some options that could be varied when connect to different servers. So when users finish connecting to one server and want to connect to another server with different options, users need to call DataBroker_SetConnectionOptions to change these options. This function could not be called when connecting. These options are called offline settings.

In AVSynchronizer, the unit for handling with one source of media is a channel. A channel is created with some settings, and during the channel is running, users could update some properties that affect the display result of this channel. Of course these settings could also be updated when the channel is not running. These settings are calling online settings. Users could call AvSynchronizer_UpdateXXChannelSettings to change to settings.

For offline settings the change would affect the substance of the connection, but the online settings affection the presentation of the channel. And the allowable calling time for these two settings is also different.