



**WIRELESS INTERNET DIAGNOSIS ANALYSIS SYSTEM
- VIDEO ON DEMAND**

COMPONENT VIEW REPORT

C++ SYNTAX

INCLUDES DOCUMENTATION

GENERATED AT AUGUST 23, 2003 (VER 2.0)

DOCUMENTED BY DONG HYUN JEONG

TABLE OF CONTENTS

CHAPTER 1. DEFINITIONS	4
MULTI (WIDAS VOD).....	4
CHAPTER 2. PROCEDURES	6
AUTOMATIC PROCESSING.....	6
SELECTIVE PROCESSING	6
REPLAYING MEDIA.....	7
IDLE MONITORING	8
UPDATING UI PARAMETERS.....	8
SHOWING MESSAGE.....	8
AUTOMATIC SYSTEM SHUT-DOWN	9
SESSION ESTABLISHMENT IN MPEG4 STREAMING SERVICE	9
DYNAMIC CHANGING CHART ITEMS.....	9
LOG DATA	10
AUTOMATIC CHANGING OUTGOING BOUND.....	11
CHAPTER 3. SOURCE CODE & EXPLANATION.....	12
CMULTIAPP.....	12
CMAINFRAME	14
CCHILDFRAME CLASSES	19
NETWORK QOS	19
APPLICATION QOS.....	19
QOS STATISTICS.....	20
CSELSSMEDIA AND CSELDSMEDIA	20
MEDIA CLASSES DERIVED FROM EACH MEDIA DLLS.....	21
CMEDIAVAROPLAYER	21
CMEDIAH264PLAYER	21
CMEDIAWAVELETPLAYER.....	22
CVAROPLAYER	22
CMWPLAYER.....	22
CHTTPPROTOCOL	22
CRASAPI.....	23
CCONTROLPORT.....	23
CPOWERSERIALPORT	23
CRfSERIALPORT	23
CSVRIPDLG	23
CVIDEOSCREENDLG.....	23
CMMSGDLG	24
CTRACEMSGDLG	24
DEFINED SDK	24
EXTRA DERIVED CLASSES	25
APPENDIX A. DEFINED PACKET	27
APPENDIX B. RAS ARCHITECTURE	29
APPENDIX C. DATA STRUCTURE.....	31
APPENDIX D. DATA FLOW DIAGRAM	33
APPENDIX E. ACCESSING THE HTTP PROTOCOL.....	37

APPENDIX F. POWER CONTROLLER & IPC.....39
APPENDIX G. PROBABILITY DENSITY FUNCTION.....40

Chapter 1. Definitions

Multi (Widas Vod)

This software is for controlling media such as streaming and downloading. Those media are playing in June (Commercial brand name pronounced by SK Telecom). The main purpose of this software is maintaining and evaluating quality of service (QoS) while using on demand services and downloading services. It is a self-operation system running automatically with initial settings. Mainly it consists of system software and system device (called power control device). We exclude brief explanation of the system device, since this document only focuses on software engineering of the system. This documentation is written for the purpose of adapting revision process easily. It consists of four parts such as Definitions, Procedures, Source code, and Appendix. Here in Definitions part, we will see what the system is and the notation method of source code. In Chapter 2. Procedures, there are more brief explanation about procedures or scenarios used in this system and also can find how the system software is designed and how it works automatically. We also can see the technical information about source code (short story) in Chapter 3. At last, we will have a look some additional information in Appendix.

<Notation method>

All variables are named using Hungarian notation method. What is the Hungarian notation? It is a variable naming convention that includes C++ information about the variable in its name (such as data type, whether it is a reference variable or a constant variable, etc).

Briefly narrow some variables here (all variables are lower case character):

Class Member variable named with prefix `m_`.

Global variable named with prefix `g_`.

DWORD : `dw`

Double or DOUBLE: `d` or `f`

float : `f`

Int or INT : `n`

bool or BOOL : `b`

TCHAR or CString : `sz` or `str`

Pointer variables : `p`

Structure : `st`

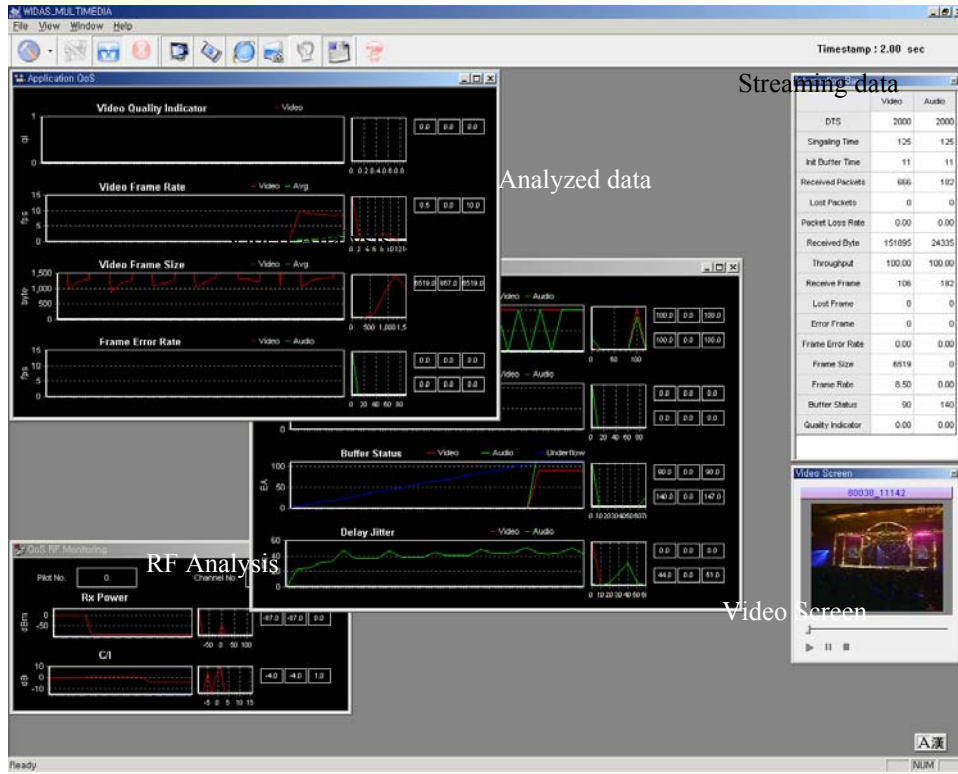
Ex) `m_pstH264` : member function, pointer and structure variable named with H264

<Cautions while reading this documentation>

Idle state or status : There are two kinds of system idle status such as waiting for user request and no action because media decoder goes to idle state.

Procedures or functions : Basically programming language consists of several procedures or functions. In this documentation two different notions denote same meaning.

WIDAS VOD 2.0 PHYSICAL VIEW REPORT



Chapter 2. Procedures

Automatic processing

As commented in CH 1., the main purpose of this software is running and evaluating QoS (Quality of Service) automatically without user's helps or commands. We call this Auto-processing. The Auto-processing denotes that if the OS or system power is shut down, the power is on by control device and operates the software to analyze media quality. We will skip the whole story about the power control device because this documentation is just for methodology of software implementation and modification. Anyway, is it possible to evaluate processing media automatically? Yes, it is. First of all, user has to define evaluating parameters to enable automatic processing. After defining some parameters, the application can detect pre-defined parameters and operates by itself even if the system re-boot or shutdown. How the application detects defined parameters? It uses a specified file including defined parameters (~\$Multi.atx). The extension of the file stands for automatic execution. Whenever the application is started, it checks existence of the file. It needs to have some extra time (twenty seconds) for checking components and changing routing table in advance.

Selective processing

Selective processing is a processing procedure that user can define parameters and select medium to process. We defined two different processing mode such as Streaming Service and Downloading Service. The former is playing a streaming media. The latter is playing downloading media. Whenever media plays, the application requests some parameters such as evaluating Quality Indicator (QI) and saving streaming data and log file to a local hard disk. Those are optional. If we want to evaluate QI, we must have an original media file to compare with streaming data. Probably you are curious to know why saving streaming data needs. It is used for replaying. We will have a look Replaying media briefly in next part.

What is the Selective processing does? Selective processing has several functions. To check system and software state, there are some monitoring procedures such as media monitoring, http monitoring, serial-port monitoring and idle status monitoring. Media monitoring checks the media status while playing a media. If some error or stop messages are occurred, it catches the messages and sends them to other functions to process. Results or error messages will be sent to web server using Hyper Text Transfer Protocol (HTTP). While sending them, UI cannot know the network connection is alive or not. If it is broken, we have to wait the protocol returns timeout message (after 15 seconds), thus http monitoring is necessary for checking the network status. With serial port communication, power control device (designed by HFR co., Ltd.) can be controlled. Sending and receiving messages synchronously between power control device and application are the key point of maintaining system. There are some unknown bugs in their applications (using the third part media decoders). If unknown bugs are occurred, the power control device will shutdown the system and re-boot operating system.

While using the third part media decoders, there is an unwanted error. Although application sends a playing request to media DLLs, the media does nothing like idle state. Whenever the application gets into the idle state, it has to be checked and managed to leave getting out of the idle status. To check the system is idle or not, we used timer function. It checks whenever time elapsed. More brief explanation described in Idle processing part.

RF data can be addressed by sending requests through serial port. In Qualcomm documentation, there are some masking method for getting data from mobile. Brief description about addressing and getting data can be found in next chapter.

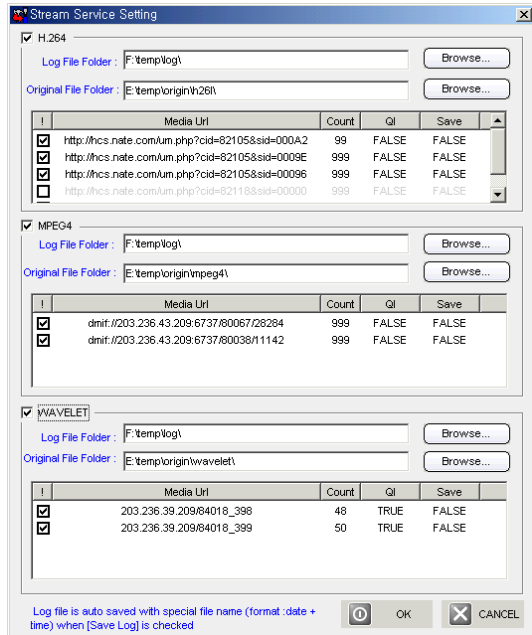


Figure 1. QoS Streaming Service Setting

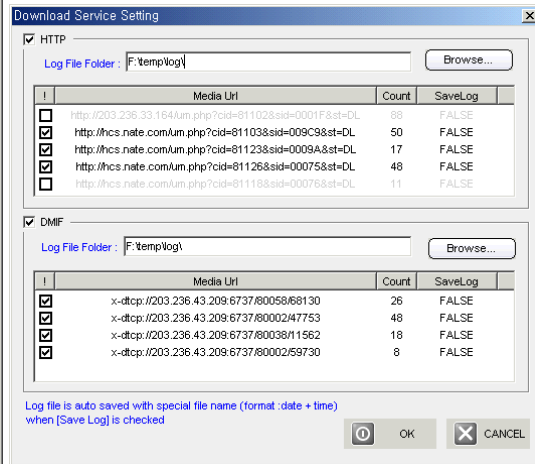


Figure 2. QoS Downloading Service Setting

As you can see the images above, there are two different setting mode such as streaming service and downloading service. Each mode has different media setting scenarios. Streaming service is for setting H.264, Mpeg4 DMIF, and Wavelet. Downloading service is for setting Mpeg4 DMIF and Mpeg4 Http (RTP). User can defined playing media and running count. Running count is for setting how many times the media will be played. If the value is above 999, it denotes endless running mode. The Qi and Save setting can be changed just double clicking with mouse.

Replaying media

As you understand it by name, the Replaying media is to replay saved data. When replaying a saved media, maintaining synchronization is important between media data and a log file which includes evaluated data. To manage synchronization efficiently, it loads evaluated data in advance. But the most serious one is that user can move the sliding bar to backward or forward position. If so, searching synchronized position in streaming data and log file needs robust time consumption. It uses pre-processing procedure to reduce the robust time consumption for searching position. Internally if user requests skipping commands (moving backward or forward), application sends a pause message with changed synchronous position, and moves the sliding bar with changed position and sends resume or play message.

In Mpeg4 DMIF (Streaming) media, it has to send stop message to media DLL when replaying is finished because of the architecture of DLL (commented by VaroVision Inc.). Note that this is a the case of streaming media not downloading media.

Idle Monitoring

As we commented shortly in Selective processing part, Idle Monitoring is indispensable for checking the system. While playing media, the application can go through to idle status intermittently. To detect this error, it uses timer function with pre-defined time value (SERVERTIMEOUT and MEDIAIDLETIME). There are two different idle checking procedures such as checking media DLLs and the application.

First, the application checks DTS time whether DTS of streaming data is updated or not. It regards the first incoming data as a start time. If the streaming data does not come out for the pre-defined time (SERVERTIMEOUT) comparing with the start time, it treats the media DLL as idle. Therefore media will be stopped and send Error message (202) to web-server.

Second, the application checks sending messages. If error or result messages are not sent to web-server for the pre-defined time (MEDIAIDLETIME), it regards as idle status; thus reset the system without sending error message to web-server.

Updating UI parameters

Decoded parameters will come out from media decoder while playing it. Each data depend on each media type; therefore updating intervals are different. Even if parameter passing interval time is defined, decoding time depends on decoders; therefore, there are two different updating modes such as active (direct) and passive updating. Passive updating means that evaluated parameters will be saved temporally in a specified memory, system acts as a substituting procedure; therefore, updating them in specified intervals (MULTI_UPDATEMONITORINGTIMER). The other one is active (direct) updating. In this mode, decoded data will be updated whenever it comes out.

Decoded data has to be changed. With raw data, user (Administrator) can not understand what the parameter means. It has to be changed to easily recognizable format. After processing, it can be shown at charts and static data boxes.

Important: Even if we defined passive mode which uses time interval, the processed data will be saved to log file following active mode pattern. In replaying mode, as motion picture and decoded data can be synchronized using DTS (data time stamp), it is hard to find exact time stamp position with altered DTS if application saves modified DTS to log file; therefore this system use direct sawing method.

Showing Message

After the system is re-booted, we can see the message that the automatic processing procedure is in process; furthermore, whenever unwanted error is occurred, the system shows the error message. All messages are for letting user (Administrator) know the system status. With the displayed message, administrator can recognize the status of the system. The messaging uses a timerfunction as an indicating process. While showing message without time (how long the message will be posted), user cannot realize whether it is acting or not. Message showing time will be decreased in a second. After the defined time is elapsed, message dialog will be disappeared by itself. The elapsed times is defined differently depends on each message attributes such as mobile reset time, automatic processing time, and etc. See more about defined time in Appendix.

Automatic system shut-down

While playing media (multi-processing mode), system will be shutdown by itself between 2:00AM ~ 2:30AM(default). It is an optional procedure. If unknown error is occurred in software or operating system, application cannot recognize the system has to be rebooted. The default value can be changed the default shutdown value in “define.h”.

Session establishment in Mpeg4 Streaming Service

Authentication procedures in Mpeg4 streaming service follows the SK Telecom policy which all authenticated information of clients saved in session database for 24 hours. After the time, all time-expired session will be removed in session database; therefore, before playing Mpeg4 streaming media, the client has to make session establishment with server if session is not established. Authentication procedure makes session establishment using http protocol (port:88) and get method.

< Mpeg4 Session duration time >

Downloading services : 30 days

Streaming services : 24 hours

< Mpeg4 Authentication procedure parameter >

<http://aaa.bbb.ccc.ddd:88/InSession.asp?MinNum=0112223333&MenuID=1234567&CID=123&SID=456&SvcType=OD>

Parameter	Comments
MinNum	Min Number
MenuID	Menu ID
CID	Content cid
SID	Content sid
SvcType	Content service type (OD/DL)
SM	SM value (optional)
SU	SU value (optional)

Dynamic Changing Chart Items

Dynamic changing chart is for en- or disabling displayed charts. It is designed for showing charts in limited space (window resolution). It has ordering and selection procedures. Ordering procedures is for changing chart order. Among the several charts, there are different priorities depend on attribute. User can define chart order by given priority. Given priorities on each parameter are not enough to display parameters (charts) on screen because of cramped space. Selection procedure is the solution for displaying some valuable parameters (charts) instead of showing all parameters.

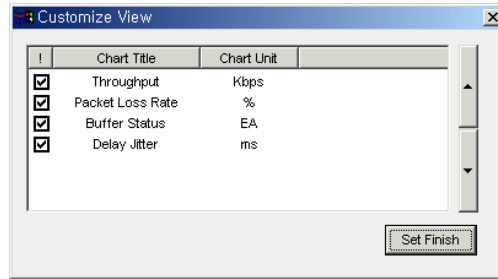


Figure 2. Customeize view(changing charblock attributes)

A chart consists of several components such as data chart, Probability Density Function (PDF) chart, legend and static boxes. We call this is a chart block. If user changes attributes, each chart block components will be affected in a module. Also each chart block has a bit different view because of different attributes.

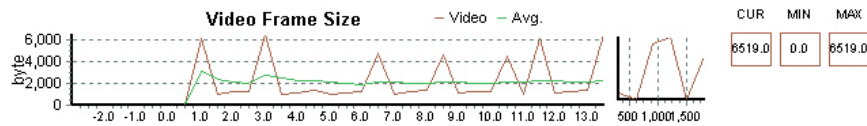


Figure 3. A chart block

Log Data

All evaluated parameters are saved in local disk when user defines saving routine. Streaming and Downloading service has different parameters. To indicate log file format and service mode, different chunk ID is defined. All log files created by this application have their own chunk ID.

Chunk Name	Chunk ID
Log File Chunk	0xA000
Parameter Data Chunk	0xA001
RF Data Chunk	0xA002
Streaming service	0x2720
Downloading service	0x271F

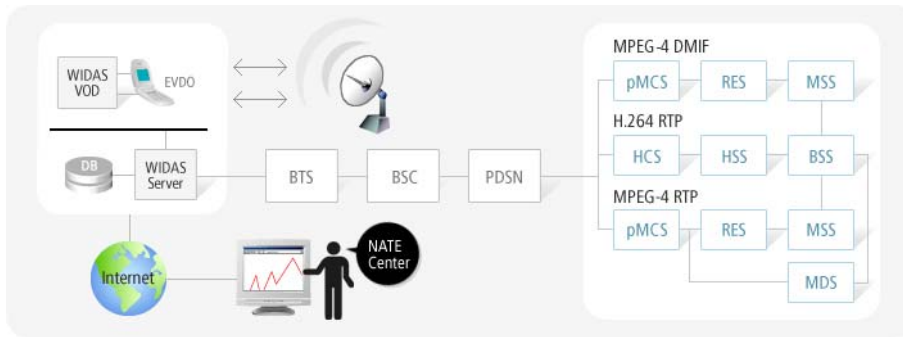
A structure of chunk has a following format:

2Byte	Dependable size
Chunk ID	Contents

Each chunk has different contents size depending on media mode (streaming or downloading). Streaming and Downloading service chunk is used for indicating media mode. Parameter Data chunk is the main chunk. It follows evaluated parameters. RF data is a bit different from parameter data. Although the parameter data is saved continuously, RF data is saved occasionally because of the characteristic of Qualcomm IC.

Automatic changing outgoing bound

Generally Internet connection uses routing table to reach a specific domain or server. In this application, there are two different Internet connections such as Ethernet and Remote Access Service using mobile. To sustain evaluation procedure using mobile, the application has to change routing table. Also to guarantee system long-life duration, routing table also has to be changed. First of all, changing routing table is for making outgoing bound to Internet using mobile. If some error occurred while connecting to media server using Remote Access Service, routing table has to be changed using local Ethernet path. If the application changes routing tables, it acts like switching device.



Chapter 3. Source code & Explanation

CMULTIAPP

```

////////////////////////////////////
CMultiApp:
See Multi.cpp for the implementation of this class
Derived from CwinApp
    
```

<Screen Changing>

Listed items used for switching screen depend on each media types. Whenever a media plays, those functions will be called. Therefore user can see different screen depends on each media.

```

CQoSApplicationChildFrm*   m_pQoSApplicationChildFrame;
CQoSApplicationView*      m_pQoSApplicationView;
CMultiDocTemplate*        m_pQoSApplicationDocTemplate;

CQoSNetworkChildFrm*     m_pQoSNetworkChildFrame;
CQoSNetworkView*        m_pQoSNetworkView;
CMultiDocTemplate*      m_pQoSNetworkDocTemplate;

CDSChildFrm*            m_pDSChildFrame;
CDSView*               m_pDSView;
CMultiDocTemplate*     m_pDSDocTemplate;

CChildFrmRf*           m_pRfChildFrame;
CQOSRFView*           m_pRfView;
CMultiDocTemplate*     m_pRFDocTemplate;

CQoSStatisticsChildFrm* m_pQoSStatisticsChildFrame;
CQoSStatisticsView*    m_pQoSStatisticsView;
CMultiDocTemplate*     m_pQoSStatisticsDocTemplate;

// temporary save used for reference item
CONPARAM               m_ConParam;
CONPARAMSTR            m_ConString;
CONPARAM               m_ConNULLParam;
CONPARAMSTR            m_ConNULLString;
DNPARAM                m_DnNULLParam;
DNPARAM                m_DnParam;
LOGFILEPARAM*         m_pLogFileParam;
LOGFILEPARAM*         m_pLogFileParamCurrentPos;
CMainFrame*            m_pMainFrame;
    
```

<Updating parameter>

Received parameter must be updated whenever data come out. Original parameter cannot be updated immediately because it has some unknown or unwanted information. It has to be refined in API function (WAPI_ProcessRealtimeData(...) and WAPI_ProcessDownloadData(...)). After finishing refining process, update function will be called. While updating refined data, the function does three kinds of things such as counting, checking and updating. To calculate average value of each parameter, it is necessary to count updating times. As I commented above, received data must be refined in API function. But RF data does not need to be refined or processed. RF data just need to have some validation process. Difference between v1.5 and v2.0 is that there is an extra routine such as temporary saving procedure while user pauses playing media. Data will be saved in linked list data structure until user stop pausing; then paused data will be updated in chart or screen immediately.

```

// update parameters
    
```

```

UINT    m_nTotal; // count used for calculate average value for each parameter
BOOL    m_bUpdateData;
CCriticalSection m_csUpdateData;
void    UpdateStreamingDataCheck(BOOL bVirtualData = FALSE);
void    UpdateStreamingData(CONPARAM *pParam, CONPARAMSTR *pString, RFPARAM *pRfParam, BOOL
bVirtualData = FALSE);
void    UpdatePausedData(UINT nMediaMode);
void    UpdateDownloadingDataCheck(BOOL bVirtualData = FALSE);
void    UpdateDownloadingData(DNPARAM *pDnParam, RFPARAM *pRfParam, BOOL bVirtualData = FALSE);
void    UpdateRfData(RFPARAM *pParam, UINT nTimeInterval);
UINT    GetTimeInterval(); // getting updating time intervals for each media

// parameter temporary saved whenever seing procedure fail
COBList    m_ObParameterList;
void        ClearObjectList();

```

<Updating parameter with interval time>

As we saw the updating procedure in Chapter 2, here are the functions that update evaluated parameters in predefined time. Here you can see the callback function (UpdateMonitoringMediaProc). While updating procedure is in progress, it checks that newly updated parameter is exists or not. If it is not updated, it updates with Null value to indicate the system is acting but media parameter does not come out.

```

UINT m_nUpdateMonitoringMediaTimer;
BOOL m_bUpdateData;
static void CALLBACK UpdateMonitoringMediaProc();
CCriticalSection m_csUpdateData;
void StartUpdateMonitoringMedia();
void StopUpdateMonitoringMedia();

```

<Getting and Setting statusbar pointer>

The Windows Forms **StatusBar** control is used on forms as an area, usually displayed at the bottom of a window, in which an application can display various kinds of status information. In this application we used to indicate predecoding status. But the problem is that media classes cannot access directly to StstuaBar; therefore we save StatusBar control temporary. With saved StatusBar control media can access StatusBar control and show predecoding status. Why can directly access to StatusBar? Media predecoding-acknowledge is sent to application from decoder (DLL) as a callback routine. In a callback function, direct accessing to main class is not possible.

```

CStatusBar* m_pStatusBar;
CStatusBar* GetStatusBar();
void SetStatusBar(CStatusBar* pStatusBar);

```

<Message dialog >

While playing each media, there are unwanted errors or indispensable operations. But how user can recognize them. To see current operations or errors, user can see through message dialog or log file. Message dialog is an optional viewing box to let user know. And the log file is for looking or finding unknown system error. Every tracing log-files saved daily in temporary directory. Also the oldest log-files will be deleted automatically. Initially log-files will be kept in two days. (TRACINGLOGDELDATE in define.h)

```

CTraceMsgDlg    m_cTraceMsgDlg; // trace msg dialog
BOOL            m_bTraceMsgDlg;
FILE*           m_pTracingLogFile; // trace logfile pointer
void            SaveTracingLog( LPCTSTR lpszItem, BOOL bSendStatus, BOOL bWebSend = TRUE);
CMsgDlg*        GetMsgDlg(); // log message dialog

```

<Initializing and changing screen >

In this application, there are several child frames included because of displaying streaming and downloading parameters. Initializing process can init several child frames and maintain current media screen status. With child frame pointers, the application can switch streaming and downloading screen and also can update real-time data dynamically.

```

// initialize child frames
void          InitDSChildFrm();
void          InitSSChildFrm();
void          InitRFChildFrm();
void          InitStatisticsChildFrm();

// hide show child frames
void          ShowDSChildFrm();
void          ShowSSChildFrm();
void          HideChildFrm(UINT nMedia);
    
```

CMAINFRAME

```

////////////////////////////////////
Derived from CMDIFrameWnd
    
```

All components in this application are divided into two groups such as managing and controlling. Managing part has some sub parts : QoS Streaming, QoS Downloading and Replaying media. While playing a media, each part can be used for controlling media status (active, stop or error occurred). QoS Streaming denotes streaming service such as Video on Demand (VOD) service. Replaying have extra functions : pause, stop and move forward or backward.

And controlling group has several thread functions and control classes such as RF Serial, Control device Serial and Http control class. Also there are a lot of flags used to check the status of each control class.

Managing groups have three processing procedures such as Auto-Processingthread, Monitoringmedia (Streaming and Downloading service) and Replay-Monitoringmedia. Those procedures always check system status while playing media. If error or event message is occurred by media components, those functions can detect and process it. For example, when error or finish message occurred, Monitoringmedia stops current playing media and start to play next media. If running count of media is zero, Monitoringmedia will stop all processing procedures and get to the idle state waiting for user request. Replay-Monitoringmedia is similar with Monitoringmedia. But the difference between two is that the Replay-Monitoringmedia checks message while playing only one time.

<Auto-Processing>

Auto-processing is to run the application automatically just after all components loaded. It reads a file "init.ini" to load RF mode and web-server IP address. And then check "~\$Multi.atx" file is exist or not. The file has setting values (User selected options). If the file exists, the application switches screen mode depending on media type. And then call each media auto-processing procedure defined in ssmedia.h or dsmedia.h.

```

CWinThread *   m_pAutoProcessingThread;
BOOL           m_bAutoProcessingEnable;// Auto processing check bit
static         UINT AutoProcessingThread(LPVOID pParam);
void           OnAutoProcessing ();
    
```

<Monitoring idle status>

While playing media, it is not possible to know that the status of media whether media component goes to idle. It has to be checked manually that media is stopped or goes to idle status. How to check the status of media? Whenever defined time is elapsed, the monitoring function checks media status. If the DTS does not updated for specified time, the status of media will be considered as idle; therefore, media will be stopped and send the pre-defined error messages (202 or 203) to web-server.

```

// Checking Media if it is working or not
UINT          m_nCurrentDts;      // used for check it is changed or not
UINT          m_nOldDts;
UINT          m_nCheckMedialDllMonitoringTimer;
void          CheckMedialDllMonitoringProc();
UINT          CheckMedialDllMonitoringStart();
    
```

```
void          CheckMedialDllMonitoringStop();
```

<HTTP controlling>

It is for sending error or result data to web-server. When some error is occurred, monitoring groups generate message to send result or error data to web-server. The media especially H.264 has authentication process. It connects to HTTP server to get redirect URL with correct authentication parameter. Authentication uses thread functions because the GUI does not know the network status (stable or not). If network is unstable, the application cannot connect to HTTP server directly. How can we know the network system is unstable? This application uses windows socket2. If network is unstable or the application cannot reach to server, callback function in socket2 sends message after 15 seconds. At this point, the application cannot do anything for 15 seconds. Certain UI resources should only be accessed on the same thread, in which they were created. Therefore while the network connecting process holds onto OS resources, other UI process cannot access OS resources. Because of the reason, authentication process is designed as a thread procedure. With the procedure, UI resources can be operated even if the network is unstable.

```
// HTTP related items
UINT          m_nGetRedirectMode;
CString       m_strMeasureStartTime;          // Media playing start time
UINT          m_nSendMsgStatus;
UINT          m_nGetRedirectUrl;
CHttpProtocol m_HttpProtocol;
CString       m_strSelectedUrl;
CString       m_strRedirectUrl;
UINT          m_nErrMsg;
BOOL          m_bErrorOccurred;
BOOL          m_bSendResultToWeb;             // Send evaluation result to web server
CThread *    m_pGetRedirectUrlThread;
CThread *    m_pSendErrorToWebThread;
CThread *    m_pSendMsgToWebThread;
BOOL         m_bSendMsgToWebThread;
BOOL         SendResultToWeb();
BOOL         SendErrorToWeb(UINT *nMsg);
static       UINT GetRedirectUrl(LPVOID pParam);
static       UINT SendErrorToWeb(LPVOID pParam);
static       UINT SendMsgToWeb(LPVOID pParam);
CString       CalcSendMsg(UINT nErrorCode = NULL);
UINT         GetHttpStatus() { return (m_nSendMsgStatus); };
void         InitHttpStatus() { m_bErrorOccurredSentMsgtoWeb = FALSE;
                               m_nSendMsgStatus = HTTPFINISHSENDDATA; m_bSendResultToWeb = TRUE; };
BOOL         WaitForMsgThread();
```

<QoS Streaming and Downloading Service>

WIDAS v1.0 and v1.5 has two processing methods such as single and multi processing. But in WIDAS v2.0 adapts hybrid method. All media information should have to be defined in a file (“initlist.ini”). With media lists, user can set some running attributes. Playing streaming service and downloading service simultaneously is not possible because each service has different attribute and displaying method. Although all controlling procedures are defined in the CMainFrame class in WIDAS v1.0 and v1.5, they are divided in two classes such as CDsMedia and CSsMedia. Each has similar processing procedures except referencing different media components.

While playing media, some unwanted errors might be occurred. When it happens, monitoring procedure catches error. Depending on error type, the application treats error messages and play next media.

Scenario of QoS downloading or streaming service:

Streaming and downloading services have a bit similar processing scenarios. Each has a lot of checking flags and processing procedures because they has to check media status and play media

continuously until user pushed stop button or system shutdown. Downloading and streaming service has different media selecting mode. When user pushed setting buttons, a dialog depending on each media called. Media type, Qi and Log files saving routine can be defined in the dialog. With defined attributes, the application checks that what kind of media type is selected and Qi or Log Save item is checked. And then, it saves setting values to a file (~\$Multi.atx) and starts monitoring in sequence. While playing media, monitoring functions are enabled for checking media statuses.

Monitoring functions are divided in several parts such as monitoring media status, making connection with mobile and checking idle state. When user canceled or stopped a playing media, monitoring function acts immediately to process next actions. Basically the application uses Internet using Remote Access Service (RAS). RAS has synchronous and asynchronous mode. This application uses asynchronous RAS connection mode. It sends connecting message through callback function. See Appendix B for more detail. Also if playing media goes to idle state, the monitoring function does the job getting out of.

```

CDsMedia m_cDsMedia;
CSsMedia m_cSsMedia;
void      MediaFinish();
BOOL      m_bPlayMedia;
void      DisconnectQoS();
BOOL      Connect();
void      InitVariables();
UINT      m_nPlayingMode;           // Realtime play or Replayingmode
static void *GetCurrentPlayingMediaAttribute(CMainFrame *pWnd);

```

<QoS Replay>

Saved log file and media file can be replayed in streaming service. While replaying media, evaluation process cannot proceed because media decoder sends out DTS value only; therefore synchronization process between saved evaluation parameter and video screen is the most important feature in replaying service. Furthermore, supporting forward and backward process while playing has more complicate procedures. Brief synchronization process will be explained in VideoScreenDlg class.

Replaying has monitoring or processing procedures as downloading or streaming service has. But the difference between them is that if there is some error occurred in replaying mode, it will stop all monitoring processes and go to the idle state for getting user request.

```

// QoS Replay related items
UINT      m_nReplaySelectedType;
CString   m_strMediaTitle;
CThread   *m_pReplayMonitoringThread;
BOOL      m_bReplayMonitoringThread;
BOOL      m_nMediaMethod;
BOOL      m_bReplayMonitoringMediaStatus;
void      ReplayMediaFinish();
static    UINT Replay_MonitoringMedia(LPVOID pParam);
void      ReplayMedia(UINT nMedia, CString strMediaFileName);
BOOL      OpenFileDialog(UINT nMode, TCHAR* szFileName);
void      OnReplayH264(CString szMediaFileName);
void      OnReplayMpeg4(CString szMediaFileName);
void      OnReplayWavelet(CString szMediaFileName);

```

<Log File>

It is used to read or write evaluated parameters in a file. As we have a look in Chapter 2, it has several chunk ID indicating the log file generated by WIDAS VOD application. A logfile name is created following time sequence (year+month+day+hour+minute+extension).

```

//      Logfile related items
BOOL      m_bFileOpen;
CFile*    m_pLogFile;
BOOL      OpenLogFile(CString strLogFileName, BOOL bMode = TRUE);
void      CloseLogFile();

```

<RF Data>

There are two kinds of RF mode in here. One is EVDO and the other is 1X mode. To get RF data, we have to send a request through serial port. The brief explanation how to get RF data can find in Qualcomm (*CDMA Dual-Mode Subscriber Station Serial Data Interface Control Document*) documentation. If we want to get EVDO RF data, sending a request is necessary. But in 1X mode, we have to send continuously if we want to get data because it acts like request-response procedures. In EVDO, is it restricted sending a message once? No. You don't need to worry about sending requests several times. We used a timer function here to send a request message. Whenever set time elapsed, timer function will send a request message through a serial port. The important thing here is that in EVDO mode RF data comes out continuously but 1X mode does not. If we want to evaluate EVDO mode RF data, we have to control update routine. This means that if RF data updates whenever it comes out, updating resource hold OS resources therefore other OS resource cannot be used. We restricted a point of updating time to the streaming data with timestamp value. Also we need to check what the most frequently displayed value is (Pilot No., Channel No.) and send them to web server because we cannot see all data on the Internet. Most process getting RF data are used or defined in WIDAS v1.0, v1.5 and v2.0. Although all procedures are defined in WIDAS v2.0, only EVDO mode is used because of changed SKT policy.

```
// RF Data
RFSTRUCT      *m_pRfPilotNoListHead;
RFSTRUCT      *m_pRfChannellListHead;
RFPARAM       m_Rfparam;
CCriticalSection m_csRfData;
BOOL          m_bRFMonitoring;
BOOL          m_bRfparamUpdate;
BOOL          m_bRFMode; // 0 : EVDO / 1 : 1x
void          InitRFData();
void          RefreshData();
void          CalcRfData(RFPARAM *pRfParam);
void          OnUpdateRealtimeRfData();
BOOL          OnRFLogRequest(); // Send RF data request
```

<Gateway Set >

When OS (Operating system) is initialized, the gateway will be set if we defined network devices. The RAS is also a kind of networking, therefore when RAS is connected, gateway (using 0.0.0.0) will be added with new host IP address. As you understand, this software is to evaluate media quality through mobile. If we did not define the initial gateway, we cannot guarantee that the media we evaluate media through Mobile or Ethernet. So we have to remove default Ethernet gateway (using 0.0.0.0) generated by initial OS networking. Moreover we have to add web-server route. The web-server will be located in LAN area, therefore using Ethernet is much more useful than using mobile connection. Typically, a packet may travel through a number of network points with routers before arriving at its destination. Routing table is used for referencing a path. Therefore if destinations are not defined in routing table, applications will use default gateway whose destination IP address might be set to 0.0.0.0.

Caution : When OS are started, routing table will be initialized in a few seconds later. In this application, automatic processing procedures has pre-defined waiting time. That is defined for adding or changing routing table effectively. It regards as a waiting procedure.

```
// Gateway
DWORD m_dwLocalIp;
DWORD m_dwWebServerIp;
CString m_strWebServerIp;
void InitGateway();
void DelInitGateway();
PMIB_IPFORWARDROW m_pIpForwardArrow;
```

Initialized Routing table (each values depend on systems)				
Interface List				
0x1	MS TCP Loopback interface			
0x3000003 ...00 e0 18 54 5a 7f	HP 10/100TX PCI Ethernet Driver			
=====				
Active Routes:				
Network Destination	Netmask	Gateway	Interface	Metric
0.0.0.0	0.0.0.0	210.115.229.1	210.115.229.124	1
210.115.229.0	255.255.255.0	210.115.229.124	210.115.229.124	1
210.115.229.124	255.255.255.255	127.0.0.1	127.0.0.1	1
210.115.229.255	255.255.255.255	210.115.229.124	210.115.229.124	1
224.0.0.0	224.0.0.0	210.115.229.124	210.115.229.124	1
255.255.255.255	255.255.255.255	210.115.229.124	210.115.229.124	1
Default Gateway: 210.115.229.1				
=====				
Persistent Routes:				
None				

<Serial Port >

To control RF data and Power control device, we used serial port data communication. In short, we had a look about how to get RF data through a serial port. Using serial port, the application can get RF data and control power control device. Power control device is the extra boundary acting while the application does nothing. If the application met unwanted error, the OS must be reset automatically to maintain the system or get long-life duration.

```
// Serial Port
BOOL      m_bRfSerialPortConnection;
BOOL      m_bPowerSerialPortConnection;
BOOL      CreateRfSerialPortConnection();
BOOL      CreatePowerSerialPortConnection();
```

<System Shutdown >

Before the system shutdown, it sends a message to power-serial port or to web-server. Therefore it has to wait for a while until sending message is finished.

```
// Shutdown program and system
BOOL      m_bWindowShutdown;
CThread   *m_pWindowShutdownThread;
static    UINT WindowShutdown(LPVOID pParam);
CString   m_strShutDownMsg; // T3 or T5
```

CCHILDFRAME CLASSES

MFC Multiple document Interface (MDI) has several child windows consisting of child frame, document and view. Listed items below are Streaming, Downloading, Application QoS, Network QoS, QoS RF and QoS Statistics classes. Each view classes are used to update streamed data. Everyone knows that what kinds of class will be used if he studied MFC; therefore we will skip some classes. Each memory pointers of each classes has temporary saved in Applilcation class (CWinApp) to reference child window.

What is a derived class? The derived class is a modified class originally defined in MFC base class to change or insert some functions. We used some derived classes to change UI forms and text color. Also we used Tee-Chart component to display real time graph and MSFlexGrid to make a possibility seeing all data at once.

NETWORK QOS

Network QoS is for evaluating media parameters relating with network state such as Throughput, Packet Loss Rate, Buffer Status and Delay Jitter.

<Calculation method >

Throughput : $((\text{numRcvPackets} - \text{numErrorPackets}) / (\text{numRcvPackets} + \text{numLostPackets})) * 100$

Packet Loss Rate : $(\text{numLostPackets} / (\text{numRcvPackets} + \text{numLostPackets} + \text{numErrorPackets})) * 100$

Buffer Status : calculated in media decoder or underflow count is calculated in this application. If buffer status is below 5, under flow count will be increased.

Delay Jitter : calculated in media decoder

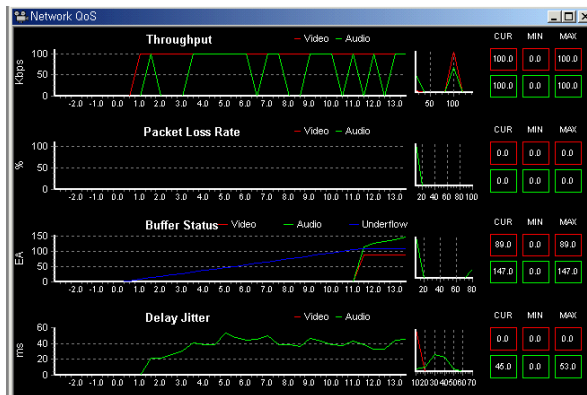


Figure 4. Network QoS

APPLICATION QOS

Application QoS is for evaluating media parameters relating with network state such as Video Quality Indicator, Video Frame Rate, Video Frame Size and Video Error Rate.

<Calculation method >

Video Quality Indicator : calculated in media decoder if checked.

Video Frame Rate : calculated in media decoder

Video Frame Size : calculated in media decoder

Video Error Rate : $(\text{numErrorFrame} / (\text{numRcvFrame} + \text{numLostFrame} + \text{numErrorFrame})) * 100$

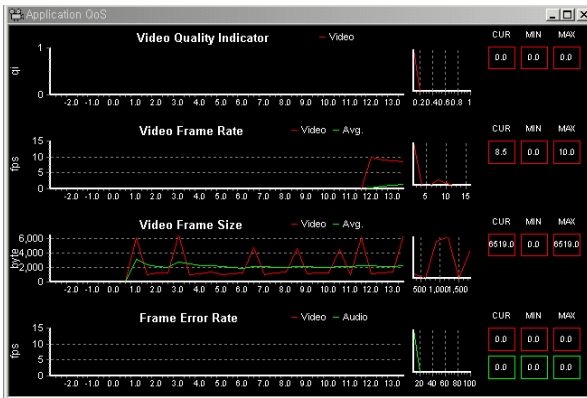


Figure 5. Application QoS

	Video	Audio
DTS	2000	2000
Singaling Time	1563	1563
Init Buffer Time	11	11
Received Packets	662	147
Lost Packets	0	0
Packet Loss Rate	0.00	0.00
Received Byte	150950	19680
Throughput	100.00	100.00
Receive Frame	106	147
Lost Frame	0	0
Error Frame	0	0
Frame Error Rate	0.00	0.00
Frame Size	6519	0
Frame Rate	8.50	0.00
Buffer Status	89	147
Quality Indicator	0.00	0.00

QOS STATISTICS

Display all original evaluation parameters out from Media decoder. Actual parameters consist of about 20 elements. Parameters displayed at statistics window depending on their priority. Least significant elements are not displayed.

CSELSSMEDIA AND CSELDSMEDIA

////////////////////////////////////

Streaming and downloading dialog for selecting media types and changing attributes

As we denoted in previous chapter, it is for selecting or changing media attributes. The key point of this class is that Logfile or Original file directories must be checked before using them because if the defined directories are illegal, media DLLs may occur some unwanted errors. Also changed attributes have to be saved in each media files for next usage.

```
// Streaming Service Construction
public:
CSELSSMedia(CWnd* pParent = NULL); // standard constructor
BOOL CheckDirectory(CString *szpPath);
void InitData(SSURLSTRUCT *pUrl, CExListCtrl *pListCtrl);

public:
CThemeHelperST m_ThemeHelper;
HICON m_hIcon;
SSURLSTRUCT* m_pstH264;
SSURLSTRUCT* m_pstMpeg4;
SSURLSTRUCT* m_pstWavelet;
BOOL m_bInitH264;
BOOL m_bInitWavelet;
BOOL m_bInitMpeg4;
void CheckH264(BOOL bEnable = FALSE);
void CheckMpeg4(BOOL bEnable = FALSE);
void CheckWavelet(BOOL bEnable = FALSE);
void GetChangedData(SSURLSTRUCT *pUrl, CExListCtrl *pListCtrl);
```

```

// Downloading Service Construction
public:
CStdMedia(CWnd* pParent = NULL); // standard constructor
DSURLSTRUCT      *m_pstHttp;
DSURLSTRUCT      *m_pstDmif;
CThemeHelperST   m_ThemeHelper;

BOOL   m_bInitHttp;
BOOL   m_bInitDmif;
public:
void InitData(DSURLSTRUCT *pUrl, CExListCtrl *pListCtrl);
void SoCheckDsdmif(BOOL bEnable = FALSE);
void SoCheckDshhttp(BOOL bEnable = FALSE);
void EnableHttpItems(BOOL bEnable = FALSE);
void EnableDmifItems(BOOL bEnable = FALSE);
void GetChangedData(DSURLSTRUCT *pUrl, CExListCtrl *pListCtrl);

```

MEDIA CLASSES DERIVED FROM EACH MEDIA DLLS

As we have a look before, there are five different kinds of media decoder or classes. Each media class includes some DLL functions or DLL classes except MPEG4 RTP. We defined some nested classes include DLL functions or DLL classes : CMediaH264Player, CMediaVaroPlayer and CMediaWaveletPlayer. MPEG4 DMIF and MPEG4 RTP have their own class and defined procedures. They are similar syntax structures but there is no extra nested class structure defined.

CMEDIAVAROPLAYER

This class is imported or exported from the VaroPlayer.dll

```

CVaroPlayer   m_Varoplayer;
BOOL          m_bPlayerInit;           // Initialization is set or not
UINT         m_nMediaStatus;          // current media status
UINT         m_nMediaErrorCode; // media error code
BOOL         m_bMediaStartBit; // media started or not
UINT         m_nMediaPlayingMode; // current media mode
BOOL         m_bPredecoding; // Predecoding enabled
void         InitPlayer(HWND hWnd, HWND hWnd_display, char *PhoneNum);
BOOL         MediaPlay(char* szURL, UINT nSizeURL, BOOL bSaveflag,
char* szSavefileName, char* szOrgfileName, CProgressCtrl *pProgress);
void         MediaPlayEnd();
void         MediaStop();
BOOL         MediaReplay(char* szFilename);
void         MediaReplayStop();
void         MediaPause(void);
void         MediaResume(void);
void         MediaMoveBegin(unsigned int timeToMove);
void         MediaMoveEnd(unsigned int timeToMove);
UINT         MediaGetContentDuration(char* szFilename);
static void  MediaDecodePutParam(unsigned int DTS, MParameter *pParam);
static void  MediaEnd(unsigned int reason);
static void  MediaPutQIProgress(unsigned int Progress);

```

CMEDIAH264PLAYER

This class is imported or exported from the win32_player_dll.dll

```

CMWPlayer   m_H264Player;
BOOL        m_bPlayerInit;           // Initialization is set or not
UINT        m_nMediaStatus;          // current media status
UINT        m_nMediaErrorCode; // media error code

```

```

    BOOL          m_bMediaStartBit; // media started or not
    UINT          m_nMediaPlayingMode; // current media mode
    BOOL          m_bPredecoding; // Predecoding enabled
    void          InitPlayer(HWND hWnd, HWND hWnd_display);
    BOOL          MediaPlay(char* szURL, UINT nSizeURL, BOOL bSaveflag,
        char* szSavefileName, char* szOrgfileName, CProgressCtrl *pProgress);
    BOOL          MediaReplay(char* pszFilename);
    void          MediaStop();
    void          MediaPause(void);
    void          MediaResume(void);
    void          MediaMoveNMP(UINT timeToMove);
    UINT          MediaGetContentDuration(char *pszFilename);
    void          MediaPlayEnd();
    static void   MediaDecodePutParam(UINT timeLastDecode, MParam *pParam);
    static void   MediaEnd(int nReason);
    static void   MediaPutQIProgress(INT Progress);

```

CMEDIAWAVELETPLAYER

This class does not have external classes. As you can see the difference between this and other media classes. There are original classes derived from DLLs such as CVaroPlayer and CMWPlayer, but here it isn't. Just some functions are exported from the TCM3QMSLib.dll

```

    BOOL          m_bPlayerInit; // Initialization is set or not
    UINT          m_nMediaStatus; // current media status
    UINT          m_nMediaErrorCode; // media error code
    BOOL          m_bMediaStartBit; // media started or not
    UINT          m_nMediaPlayingMode; // current media mode
    BOOL          m_bPredecoding; // Predecoding enabled
    void          InitPlayer(HWND hWnd, HWND hWnd_display, char *PhoneNum);
    BOOL          MediaPlay(char* szURL, UINT nSizeURL, BOOL bSaveflag,
        char* szSavefileName, char* szOrgfileName, CProgressCtrl *pProgress);
    void          MediaPlayEnd();
    void          MediaStop();
    BOOL          MediaReplay(char* szFilename);
    void          MediaReplayStop();
    void          MediaPause(void);
    void          MediaResume(void);
    void          MediaMoveBegin(unsigned int timeToMove);
    void          MediaMoveEnd(unsigned int timeToMove);
    UINT          MediaGetContentDuration(char* szFilename);
    static void   MediaDecodePutParam(unsigned int DTS, MParameter *pParam);
    static void   MediaEnd(unsigned int reason);
    static void   MediaPutQIProgress(unsigned int Progress);

```

CVAROPLAYER

This class is exported from the VaroPlayer.dll
Mpeg4 Media defined by Component company

CMWPLAYER

This class is exported from the win32_player_dll.dll
H264 Media defined by Component company

CHTTPPROTOCOL

```

////////////////////////////////////
CHttpProtocol command target

```

To send processed data or error to web-server, we use this. Also H.264 media has redirecting url method using this class. There are two kinds of methods in HTTP such as Get and Post method. We tested the methods both and decided to send message using Get method. When we use Post method, we found that HTTP parser page didn't recognize sometimes (We don't know why?).

```
BOOL HttpConnectionCheck();
BOOL HttpConnection(CString szUrl, CString szHeaders);
BOOL HttpGetMessage(TCHAR *pszMsgResult, TCHAR *pszMsg);
BOOL HttpSendMessage(TCHAR *pszMsgResult, TCHAR *pszPostData);
BOOL HttpMakeConnection(LPCTSTR szAddress, LPCTSTR szHeaders);
BOOL HttpPostMessage(TCHAR *strRcvValue, LPCTSTR szAddress, LPCTSTR szHeaders,
    char *szPhoneNum, TCHAR *szPostData, UINT nMode, UINT nMethod );
static UINT MsgPostingThread(LPVOID pParam);
```

CRASAPI

CCONTROLPORT

```
////////////////////////////////////
CControlPort window of CRasAPI
```

To use RAS API, we have to define some arguments. This class is for communicating with RAS API class.

```
DWORD InitRasAPI(LPCTSTR pstrStatus, LPCTSTR pstrModemName,
    LPCTSTR pstrPasswd, LPCTSTR pstrPhoneNum,
    LPCTSTR pstrUser, LPCSTR pstrEntry);
DWORD DialUpNetwork(LPVOID lpvNotifier = NULL);
BOOL DialHangUp();
BOOL ConnectRasAPI(LPVOID lpvNotifier = NULL);
BOOL DisconnectRasAPI();
```

CPOWERSERIALPORT

```
////////////////////////////////////
Controlling device
```

CRFSERIALPORT

CSVRIPLDLG

```
////////////////////////////////////
CSvrIpDlg dialog
Derived from Cdialog
Local saved file can be loaded (ip.txt) here.
```

CVIDEOSCREENDLG

```
////////////////////////////////////
CVideoScreenDlg dialog
```

While playing media, we can see video screen and catch the status of streaming data. There two different mode in here. One is real-time playing media and the other is replaying media. As you can see the screen, there are several controls: title, screen, slide bar and several buttons. Buttons are used when we replay media (Play or Resume, Pause and Stop).

```
HWND GetWndScreenHandle();
void ShowVideoScreen(UINT nScreen ,UINT nMode, CString strTitle);
void EnableButtons(BOOL bStatus);
void PlayMedia(UINT nMediaType = NULL, BOOL bFlag = FALSE,
               UINT nDurationTime = NULL);
void UpdateMediaTime(UINT nDurationTime);
```

CMMSGDLG

```
////////////////////////////////////
CMsgDlg dialog
```

How can user monitor the software and catch error? There is no solution to recognize error or progressing status. To display error or specific message to screen, we use this class to show error or message. Displaying time is defined by timer function. After defined time collapsed, window will hide itself.

```
void ShowWindowView(int nCmdShow, UINT nTime = NULL);
void ShowError(UINT nErrorCode, UINT nMode);
void ShowMsg(UINT nMode, UINT nMsgCode, CString strFileName = _T(""));
void HideMsgWindow();
```

CTRACEMSGDLG

```
////////////////////////////////////
CMsgDlg dialog
```

Used for showing and updating log messages.

```
void AddString(LPCTSTR lpszItem );
void ShowError(UINT nErrorCode, UINT nMode);
```

DEFINED SDK

```
////////////////////////////////////
Some frequently used functions are defined as SDK procedures.
```

```
[WndApi_Wnd.h]
Windows related items.
void WINAPI WAPI_ShutDownWnd();
CString WINAPI WAPI_GetItemText(CWnd *pWnd, UINT nID);
void WINAPI WAPI_FlexGridSetSellData(CMSFlexGrid *pFlexGrid, long Row, long Col,
                                     CString strData, long nColWidth,
                                     short nAlign = flexAlignCenterCenter);
```

```
[WndApi_Data.h]
Streaming data or logfile data related items.
CString WINAPI WAPI_GetModulePath();
BOOL WINAPI WAPI_DeleteFile(CString strFilename);
BOOL WINAPI WAPI_ReadWriteInitFile(CString strFilename, UINT nMode, BOOL &bRFMode,
                                    CString *pstrWebServerIp, CString *pstrLocalIp = NULL,
                                    DWORD *pWebServerIp = NULL, DWORD *pLocalIp = NULL);
```

```

BOOL WINAPI WAPI_ReadWriteAutoProcessFile(PROCESSMULTI *stData,
    CString strFilename, UINT nMode);
void WINAPI WAPI_ReadUrlFile(URLSTRUCT *stUrl, CString strFilename);
void WINAPI WAPI_InitSet(MParam* pOldParam, MParam* pCurParam = NULL);
void WINAPI WAPI_CalculateRcvData(CONPARAM *pCONParam, MParam *pMParam,
    MParam *pOldMParam);
void WINAPI WAPI_ConvertRcvDatatoString(CONPARAM *pCONParam,
    CONPARAMSTR *pCONString);
void WINAPI WAPI_ProcessRealtimeData(CONPARAM *pCONParam,
    CONPARAMSTR *pCONString, MParam *pMParam, MParam *pOldMParam);
void WINAPI WAPI_ProcessSavedData(UINT nDts);
void WINAPI WAPI_ProcessSavedData(UINT nDts, CONPARAM *pCONParam,
    CONPARAMSTR *pCONString, MParam *pMParam);
void WINAPI WAPI_ProcessRfData(RFPARAM *pRFPParam);
void WINAPI WAPI_SearchCurrentSavedDataPos(UINT nDts);
BOOL WINAPI WAPI_SveRcvData(CONPARAM *pCONParam);
BOOL WINAPI WAPI_ReadSveData(CONPARAM *pCONParam);
void WINAPI WAPI_FinishPlaying(UINT nMode);

```

```

void WINAPI WAPI_ReadSvedData();
void WINAPI WAPI_DeleteUploadedData();
BOOL WINAPI WAPI_ReadGatewayFile(PMIB_IPFORWARDROW pArrow);
    BOOL WINAPI WAPI_WriteGatewayFile(PMIB_IPFORWARDROW pArrow);

```

[WndApi_Chart.h]
Chart related items.

```

void WINAPI WAPI_InitChartData(CWnd *pWnd, DOUBLE nCur,
    UINT ID_CUR, double *pdCur, UINT ID_MAX, double *pdMax,
    UINT ID_MIN, double *pdMin, UINT ID_AVE, double *pdAve);
void WINAPI WAPI_InitChartData(CWnd *pWnd, FLOAT nCur,
    UINT ID_CUR, FLOAT *pdCur, UINT ID_MAX, FLOAT *pdMax,
    UINT ID_MIN, FLOAT *pdMin, UINT ID_AVE, FLOAT *pdAve);
void WINAPI WAPI_InitChartData(CWnd *pWnd, UINT nCur,
    UINT ID_CUR, UINT *pdCur, UINT ID_MAX, UINT *pdMax,
    UINT ID_MIN, UINT *pdMin, UINT ID_AVE, UINT *pdAve);
void WINAPI WAPI_InitChartData(CWnd *pWnd, INT nCur,
    UINT ID_CUR, INT *pdCur, UINT ID_MAX, INT *pdMax,
    UINT ID_MIN, INT *pdMin, UINT ID_AVE, INT *pdAve);
void WINAPI WAPI_UpdateChartData(CWnd *pWnd, UINT nTotal, DOUBLE nCur,
    UINT ID_CUR, double *pdCur, UINT ID_MAX, double *pdMax,
    UINT ID_MIN, double *pdMin, UINT ID_AVE, double *pdAve);
void WINAPI WAPI_UpdateChartData(CWnd *pWnd, UINT nTotal, FLOAT nCur,
    UINT ID_CUR, FLOAT *pdCur, UINT ID_MAX, FLOAT *pdMax,
    UINT ID_MIN, FLOAT *pdMin, UINT ID_AVE, FLOAT *pdAve);
void WINAPI WAPI_UpdateChartData(CWnd *pWnd, UINT nTotal, UINT nCur,
    UINT ID_CUR, UINT *pdCur, UINT ID_MAX, UINT *pdMax,
    UINT ID_MIN, UINT *pdMin, UINT ID_AVE, UINT *pdAve);
void WINAPI WAPI_UpdateChartData(CWnd *pWnd, INT nTotal, INT nCur,
    UINT ID_CUR, INT *pdCur, UINT ID_MAX, INT *pdMax,
    UINT ID_MIN, INT *pdMin, UINT ID_AVE, INT *pdAve);

```

EXTRA DERIVED CLASSES

////////////////////////////////////

While using MFC in VC++, you can define derived classes to modify or change more detail attributes such as changing Form color or UI design and supporting dynamic changing mode. Some useful examples can be found on the Internet (<http://www.codeguru.com>). Most of important derived class in this application is dynamic changing chart display (CChartControl class). The class is not actual derived class but user defined class. But it is a lot of job working class.

Dynamic chart supports changing screen position and size automatically whenever user changes window size. But some attributes are fixed in this application; therefore not all child windows resized in same ratio. As you can see Figure 6, when windows size is changed, PDF and static box size does not exactly changed in same ratio with other windows size.

WIDAS VOD 2.0 PHYSICAL VIEW REPORT

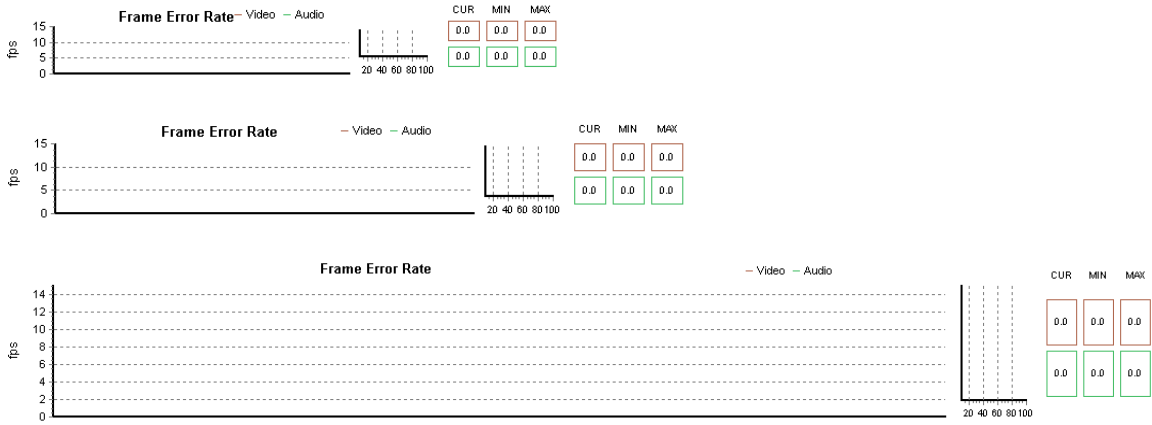


Figure 6. Dynamic resizing

APPENDIX A. DEFINED PACKET

Used Protocol is HTTP(Hyper Text Transfer Protocol). There are two kinds of method to send data using HTTP such as POST and GET method. We here used GET method. A data packet consists of several arguments. Each argument is divided by separator (minus character "-"). All data are converted to character type before send. If a value in a data packet is not exists, we just send character "F".

Error Code :

- 0 : Finished Media
- 100 : File open error
- 101 : File processing error
- 150 : Cannot Connect to Media Server
- 200 : Streaming server connecting error
- 201 : Streaming data processing error
- 202 : Streaming data does not comes out (Received at least one packet)
- 203 : Streaming data does not comes out
- 500 : RAS connecting error
- 501 : Reset Mobile (cannot make a call)
- 502 : Reset Mobile (RF data does not come out)
- 900 : Internal error

Data Packet Arguments :

Streaming Service

- 1 : Network ID (1:EVDO, 2:1x)
- 2 : Codec Type(MPEG4_DMIF:1,MPEG4_RTP:2,H.264_RTP:3,Wavelet:4)
- 3 : Service Type (1:Streaming, 2:Download)
- 4 : Media server IP(FFFFFFFF)
- 5 : CP Identifier(CID)
- 6 : Service Identifier(SID)
- 7 : Meida Device IDID(#1-#99 : fix)
- 8 : Measure Success/Fail (1:Success, 2:Fail)
- 9 : Fail Reason [1]
- 10 : Measure Start Time(YYYYMMDDHHMMSS)
- 11 : Measure Duration Time(MMSS)
- 12 : Signalling Time(ms)
- 13 : Initial Buffering Time (ms)
- 14 : Average Video Packet Loss Rate(%)
- 15 : Average Video Rate (KBit Per Second)
- 16 : Average Video Frame Delivery Time(ms)
- 17 : Average Video Quality Indicator(0-1)
- 18 : Average Video Perceptual Quality Indicator(1-5)
- 19 : Average Video Delay Jitter
- 20 : Average Video Frame Size(byte)
- 21 : Average Video Frame Error Rate(%)
- 22 : Average Video Frame Rate(fps)
- 23 : Average Video Buffer Status(EA)
- 24 : Video Buffer Underflow Count(EA)
- 25 : Average Audio Packet Loss Rate(%)
- 26 : Average Audio Rate (KBit Per Second)
- 27 : Average Audio Frame Delivery Time(ms)
- 28 : Average Audio Quality Indicator(0-1)
- 29 : Average Audio Delay Jitter
- 30 : Average Audio Frame Size(byte)
- 31 : Average Audio Frame Error Rate(%)

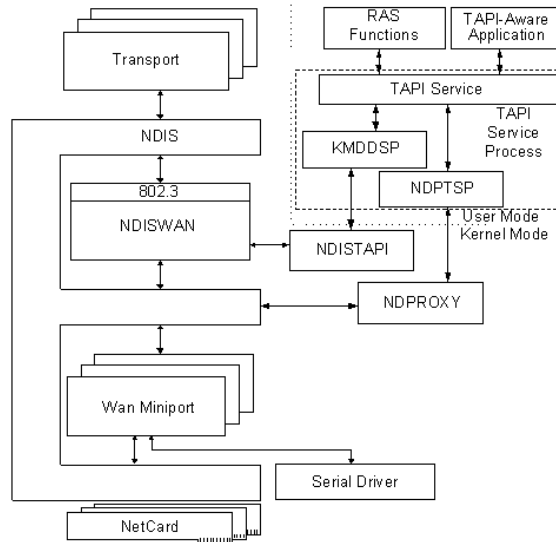
32 : Average Audio Frame Rate(fps)
33 : Average Audio Buffer Status(EA)
34 : Audio Buffer Underflow Count(EA)
35 : The most frequently measured Pilot No
36 : The most frequently measured Channel No
37 : Average Rx Power
38 : Average C/I

Downloading Service

12 : Signalling Time(ms)
13 : Average Round Trip Delay(For RTP)
14 : Average Data Block Arrival Interval
15 : Average Throughput(bps)
16 : Average TCP Throughput(bps)
17 : The most frequently measured Pilot No
18 : The most frequently measured Channel No
19 : Average Rx Power
20 : Average C/I

APPENDIX B. RAS ARCHITECTURE

This shows the relationship between Remote Access Service (RAS) and how drivers for wide area network cards such as ISDN, X.25, and Switched 56 adapters use the services of NDIS and NDISWAN to communicate in both standard WAN and connection-oriented WAN environments. The following figure shows the RAS architecture.



RAS Architecture

The following describes the NDISWAN intermediate NDIS driver, the NDISTAPI driver, and the NDPROXY driver and provides a more detailed view of the entire RAS system.

The components of WAN, RAS, and TAPI that are shown in the preceding figures are described next.

RAS Functions

The RAS set of functions allows user-mode applications to make RAS connections. After a RAS connection is established, applications can connect to network services using standard network interfaces such as Windows Sockets, NetBIOS, Named Pipes, or RPC.

Transports

The RAS system component provides transports such as PPP Authentication (PAP, CHAP) and network configuration protocols (IPCP, IPXCP, NBFCP, LCP, and so forth). A WAN miniport driver implements only PPP media-specific framing.

TAPI Service

The TAPI service (*tapisrv.exe*) presents the Telephony Service Provider Interface (TSPI) of different service providers to TAPI-aware applications. Applications use specific service providers to communicate with specific device types. These service providers are DLLs that run in the context of the TAPI service process. The operating system supplies service providers that both standard and CoNDIS WAN miniport drivers can use to communicate with user-mode applications.

KMDDSP

This component is a service provider DLL that runs in the context of the TAPI service process. The *kmddsp.tsp* component presents a TSPI interface to TAPI-aware applications so that NDISTAPI can communicate with user-mode applications. This component converts user-mode requests to corresponding TAPI OIDs for NDISTAPI.

NDISTAPI

This component implements TAPI with a kernel-mode portion of the TAPI interface. The *ndistapi.sys* component communicates with standard WAN miniport drivers by routing TAPI-related OID requests with the **NdisRequest** function to the appropriate standard WAN miniport driver.

NDPTSP

This component is a service provider DLL that runs in the context of the TAPI service process. The *ndptsp.tsp* component presents a TSPI interface to TAPI-aware applications so that NDPROXY can

communication with user-mode applications. This component converts user-mode requests to corresponding TAPI-CO-related OIDs for NDPROXY.

NDPROXY

This component implements TAPI by encapsulating TAPI parameters in NDIS structures when making and accepting calls. The *ndproxy.sys* component communicates with TAPI through the TSPI interface of NDPTSP. The *ndproxy.sys* component communicates through NDIS with NDISWAN and a CoNDIS WAN NIC miniport driver. This component presents a client interface to a miniport driver and a call manager interface to NDISWAN. NDISWAN presents a client interface to this component. A miniport driver presents a call manager interface to this component. This component enumerates TAPI capability of a CoNDIS WAN miniport driver by calling the **NdisCoRequest** function with TAPI-CO-related OIDs. This component also registers the TAPI-specific address family, creates virtual connections (VCs), makes and accepts calls, and activates VCs so that data can be sent and received on those VCs.

NDISWAN

The NDISWAN intermediate NDIS driver supports PPP protocol/link framing, compression, and encryption. NDISWAN supports both standard and connection-oriented miniport drivers. The *ndiswan.sys* driver communicates with standard WAN NIC drivers through two interfaces:

NDIS WAN interface

NDIS miniport driver interface

The *ndiswan.sys* driver communicates with CoNDIS WAN miniport drivers through the connection-oriented miniport driver interface.

Serial Driver

This component is a standard device driver for internal serial ports or multiport serial cards. The built-in asynchronous WAN miniport driver for Windows® 2000 and later uses the internal serial driver for modem communications. Any driver that exports the same functions as the serial driver will work with the built-in asynchronous WAN miniport driver.

X.25 vendors can choose to implement serial driver emulators for the X.25 card. In this case, each virtual circuit on the X.25 card appears as a serial port (with an X.25 PAD attached to it). The connection interface must correctly emulate serial signals such as DTR, DCD, CTS, RTS, and DSR.

X.25 vendors who choose to implement a serial driver emulator for their X.25 card must also make an entry for their PAD in the *pad.inf* file. This file contains the command/response script needed to make a connection through the X.25 PAD. For more information about the *pad.inf* file, see the Remote Access Service in the Platform SDK.

WAN Miniport Driver

Depending on environment, ISDN, Switched 56, and X.25 vendors should write either a standard or CoNDIS WAN miniport driver for their NICs.

Built on Thursday, July 19, 2001 exerted from MSDN

APPENDIX C. DATA STRUCTURE

Original streaming data (structure name is Mparameter) must be calculated to use in our UI. Here CONPARAM structure shown below are used to save temporary. Some calculate formulas are commented right side of value name.

```

untimeLastDecode_A;      // DTS(timestamp)
untimeLastDecode_V;

// 1. Signaling Time
unSignalingTime_A;
unSignalingTime_V;      // = timeSignalDESC + timeSignalSETUP +
                        // timeSignalPLAYwait

// 2. Packet Loss Rate
dPacketLossRate_V;
dPacketLossRate_A;      // = ( numLostPackets /
                        // (numRcvPackets + numLostPackets + numErrorPackets))
                        // Receive Packets

unRcvPackets_A;
unRcvPackets_V;
unLostPackets_A;        // Lost Packets
unLostPackets_V;
unErrorPackets_A;      // Error Packets
unErrorPackets_V;

// 3. Bit Rate
dCurrentBitRate_A;
dCurrentBitRate_V;      // = byteReceived_v / timeLastDecode_v
unReceivedByte_A;
unReceivedByte_V;

// 4. Frame Delivery Time(Buffering Time)
unFrameDeliveryTime_A;
unFrameDeliveryTime_V;

// 5. QualityIndicator
fQualityIndicator_A;
fQualityIndicator_V;

// 6. CurrentJitter
unDelayJitter_A;
unDelayJitter_V;

// 7. Throughput
dThroughput_A;
dThroughput_V;          // = ( numRcvPackets - numErrorPackets ) /
                        // ( numRcvPackets + numLostPackets)

// 8. CurrentMaxFrameSize
unFrameSize_A;
unFrameSize_V;

// 9. Frame Error Rate
dFrameErrorRate_A;
dFrameErrorRate_V;      // = numErrorFrame / (numRcvFrame + numLostFrame + numErrorFrame)
unRcvFrame_A;
unRcvFrame_V;
unLostFrame_A;
unLostFrame_V;
unErrorFrame_A;
unErrorFrame_V;

```

WIDAS VOD 2.0 PHYSICAL VIEW REPORT

```
// 10. Retransmission Ratio
unRetRequest_A;
unRetRequest_V;
unRetSuccess_A;
unRetSuccess_V;
dRetransmissionRatio_A; // = numRetRequest_a / numRetSuccess_a * 100.
dRetransmissionRatio_V; // = numRetRequest_v / numRetSuccess_v * 100.

// 11. Initial Buffering Time
unInitBufferingTime_A;
unInitBufferingTime_V;

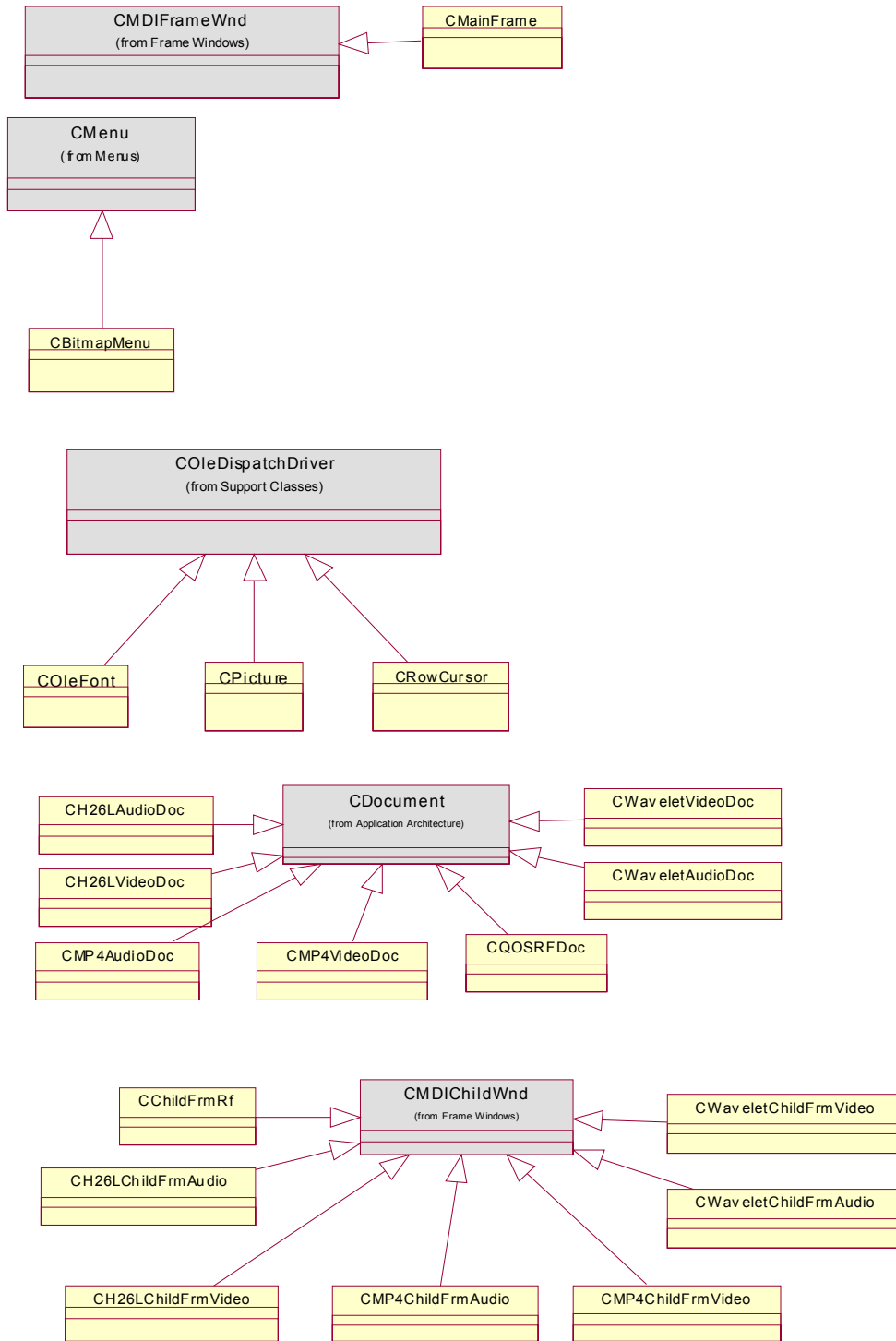
// 12. Header Size
unPacketHeaderSize_A;
unPacketHeaderSize_V;

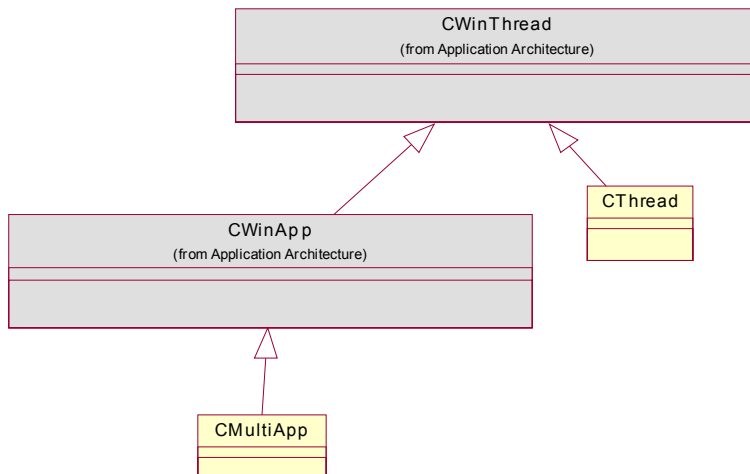
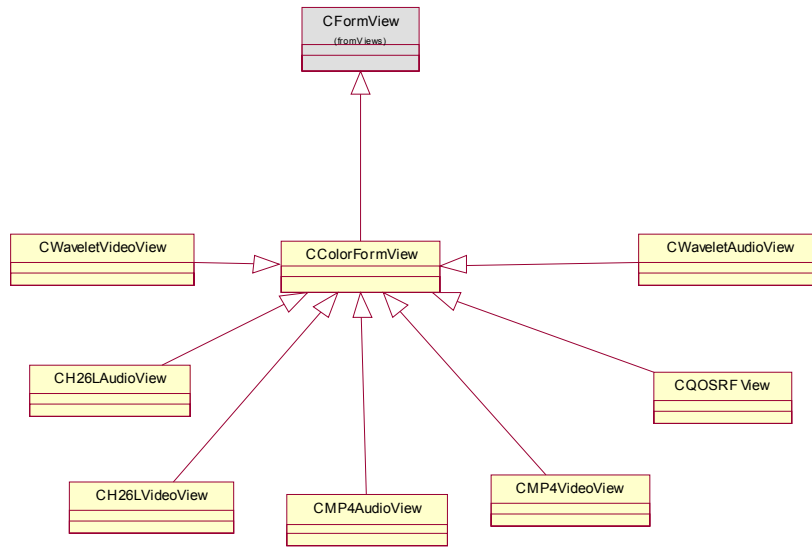
// 13. Original Frame Rate
fOriginFrame_A;
fOriginFrame_V;
fFrameRate_A; // Displayed video frame per second
               = unDecodedAU_A / untimeLastDecode_A * 1000 (sec)
fFrameRate_V; // Displayed Audio frame per second
               = unDecodedAU_V / untimeLastDecode_V * 1000 (sec)

unBufferStatus_A; // Buffer status
unBufferStatus_V;

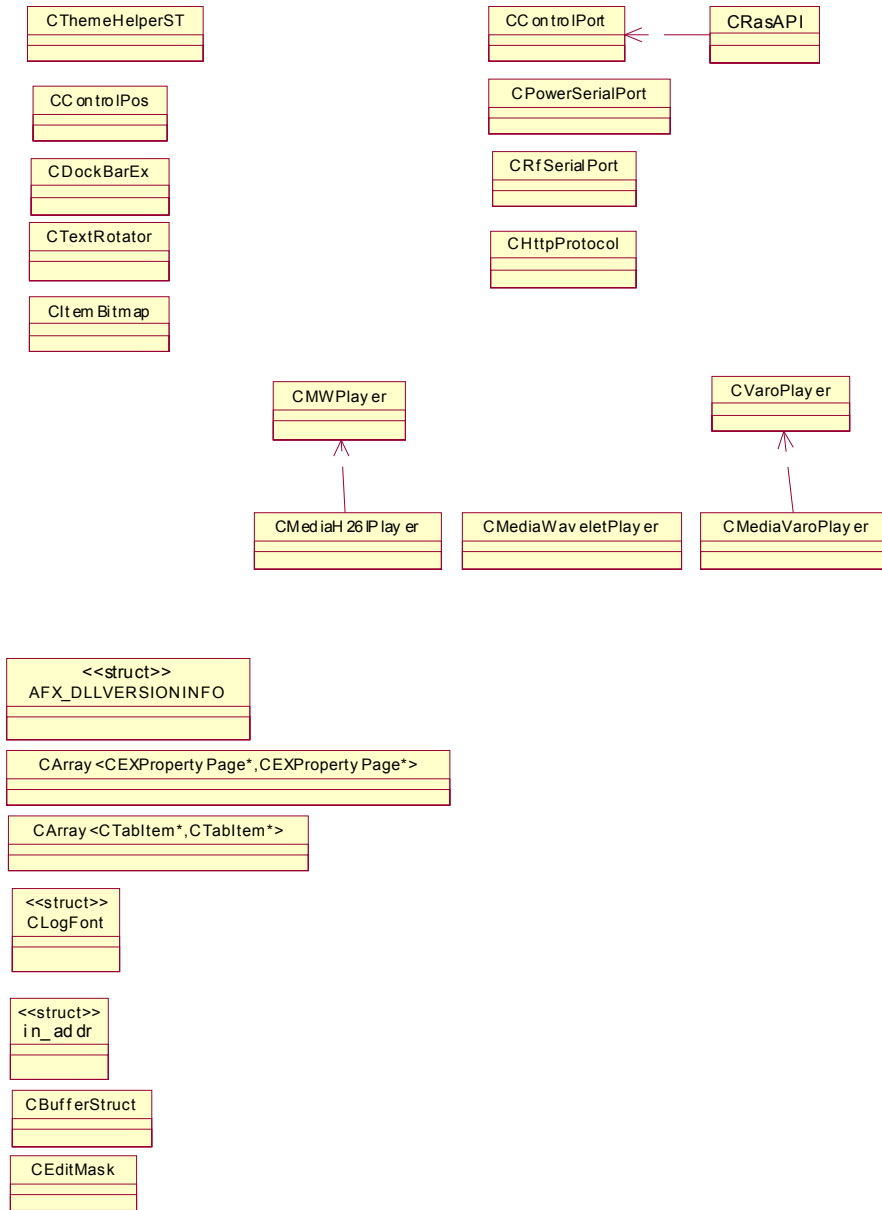
char* DIBInfo;
```

APPENDIX D. DATA FLOW DIAGRAM



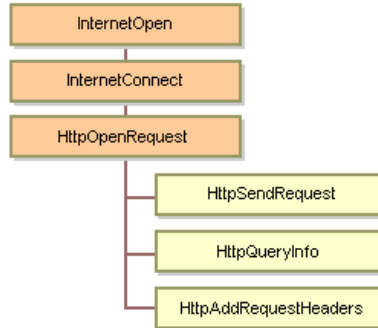


WIDAS VOD 2.0 PHYSICAL VIEW REPORT



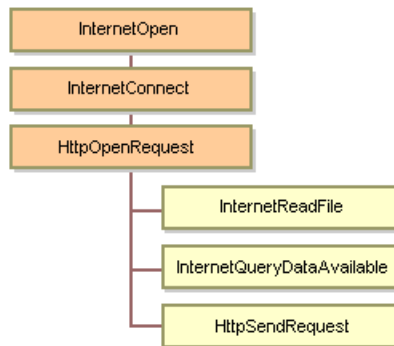
APPENDIX E. ACCESSING THE HTTP PROTOCOL

Use the HTTP functions provided by WinInet to use the HTTP protocol to access resources on the Internet. The following illustration shows the relationships of the WinInet functions used to access the HTTP protocol. Shaded boxes represent functions that return **HINTERNET** handles, while the plain boxes represent functions that use the **HINTERNET** handle created by the function on which they depend.



HttpAddRequestHeaders, **HttpQueryInfo**, and **HttpSendRequest**, are dependent on the **HINTERNET** handle created by **HttpOpenRequest**.

The following illustration shows the WinInet functions that use the **HINTERNET** handle created by **HttpOpenRequest** after it is sent by **HttpSendRequest**. The shaded boxes represent functions that return **HINTERNET** handles, while the plain boxes represent functions that use the **HINTERNET** handle created by the function on which they depend.



After **HttpSendRequest** has been used on the handle returned by **HttpOpenRequest**, **InternetQueryDataAvailable**, and **InternetReadFile**, can be used on that handle.

To use the HTTP WinInet functions

Call the **InternetOpen** function to initialize an Internet handle.

InternetOpen creates the root **HINTERNET** handle used to establish the HTTP session. The **HINTERNET** is used by all subsequent functions.

Call **InternetConnect** using the **HINTERNET** returned by **InternetOpen** to create an HTTP session.

When calling **InternetConnect**, specify **INTERNET_DEFAULT_HTTP** for the *nServerPort* parameter and **INTERNET_SERVICE_HTTP** for the *dwService* parameter.

InternetConnect uses the handle returned by **InternetOpen** to create a specific HTTP session. **InternetConnect** initializes an HTTP session for the specified site, using the arguments passed to it and creates **HINTERNET** that is a branch off the root handle. **InternetConnect** does not attempt to access or establish a connection to the specified site.

Call **HttpOpenRequest** to open an HTTP request handle.

HttpOpenRequest uses the handle created by **InternetConnect** to establish a connection to the specified site.

Call **HttpSendRequest** using the handle created by the **HttpOpenRequest** to send an HTTP request to the HTTP server.

Call **InternetReadFile** to download data.

–Or–

Call `InternetQueryDataAvailable` to query how much data is available to be read by a subsequent call to **InternetReadFile**.

Call `InternetCloseHandle` to close the handle created by **HttpOpenRequest**.

Call **InternetCloseHandle** to close the HTTP session created by **InternetConnect**.

Call **InternetCloseHandle** to close the handle created by **InternetOpen**.

APPENDIX F. POWER CONTROLLER & IPC

To check whether the system is operating or not, power controller and the software system must communicate each other. Here is the message to communicate.

T0 : Initialize controller	Response : R0
T1 : ACK to indicate software is acting correctly	Response : R1
T2 : Reset Mobile	Response : R2
T3 : System reset	Response : R3
T4 : IPC turned off	Response : R4*
T5 : System shutdown	Response : R5

As you can see above list, prefix T- indicate newly occurred message and prefix R- is response message. All messages (T-) get through to the power controller except T4. T4 is occurred in power controller when the administrator turn off the system and send it to the system software indicating power controller will shutdown OS in 30 seconds.

*) This message does not come out from the power controller. It must be sent to power controller by the system application.

APPENDIX G. PROBABILITY DENSITY FUNCTION

Probability distributions are typically defined in terms of the probability density function. However, there are a number of probability functions used in applications. We will have a look only about Probability Density Function used in this application.

For a continuous function, the probability density function (PDF) is the probability that the variate has the value x . Since for continuous distributions the probability at a single point is zero, this is often expressed in terms of an integral between two points.

$$\int_a^b f(x)dx = \Pr[a \leq X \leq b]$$

For a discrete distribution, the PDF is the probability that the variate takes the value x .

$$f(x) = \Pr[X = x]$$

The following is the plot of the normal probability density function.

