

WatchTVPro Ex 3.x Plugin API Documentation

initial version 1.03
01.12.2007

Any WatchTVPro Ex plugin is a regular Dynamic Link Library (DLL) and shall be placed into the application's plugins folder to be loaded at program start.

Parts of the used plugin data structures coming out of the public domain to achieve some compatibility with existing plugins. Some other functions are proprietary WTV functions. You should consider which main application will be compatible with your plugin. The following documentations are referring to WatchTVPro Ex Version 3.01 and above only!

A plugin mainly do the following things:

- A plugin may response to certain events which are received by the main application
- A plugin may send commands to the application in order to achieve its purpose
- A plugin may do something without touching the main application
- A plugin may provide a pull down menu

The plugin will receive events, by exporting regular functions with a given function prototype. WTV will call this function(s) if an event appears.

The following function will be called to recognize the DLL as a compatible plugin:

This function must be implemented!

```
typedef VOID (*On_Send_Dll_ID_Name)(char *Name );
```

- Name is a pointer to a character array with fixed size of 128 bytes

You **MUST** copy your plugin name to this location. The given string names the plugin and also will be used to append the plugin pull-down menu inside WTV.

The following function will be called to initialize the plugin:

This function must be implemented!

```
typedef VOID (*On_Start)(HINSTANCE , HWND , BOOL , INT , UCHAR*HotKey,  
UCHAR*Vers, int *ReturnValue );
```

- hInstance is the instance handle of WTV
- hWnd is the window handle of WTV
- Boolean is MD legacy/reserved (not used by WTV)
- Integer is your plugin id index number (you may need this by using filter functions)
- UCHAR* is MD legacy/reserved (not used by WTV)
- UCHAR* Pointer to a zero terminated character array (version string)
- int is MD legacy/reserved (not used by WTV)

You may init your plugin when this function is called. You may store the hInstance, hWnd, plugin_id and/or version string for later use.

The following function will be called to finish the plugin:

This function must be implemented!

```
typedef VOID (*On_Exit)(HINSTANCE , HWND , BOOL);
```

- hInstance is the instance handle of WTV
- hWnd is the window handle of WTV
- Boolean is MD legacy/reserved (not used by WTV)

You may free all of your resources here and finish all of your thread(s) etc. . Further you should stop sending any window messages to the main application window. **Important:** Your plugin DLL will be unloaded shortly after this call. Return only when you know, that you finished your work!

On any channel change inside WTV the following function will be called:

Optional:

```
typedef VOID (*On_Channel_Change)(struct TProgramm TP );
```

- TP is a data structure (see annex A)

You will receive reception data of the currently active channel (Name, PIDs, Ids, FEC, Frq, Symbolrate and so on). See Annex A for type definition.

The following function will be called when WTV received a WM_COMMAND message.

Optional:

```
typedef VOID (*On_Menu_Select)( int MenuID );
```

- MenuID is the command identifier of the called command

You will receive any WM_COMMAND message(s) of the WTV main window. Of course, you will also receive commands of your plugins pull down menu, which is the main purpose of this event.

Your plugin may receive the Event Information Table by exporting the following function.

Optional:

```
typedef VOID (*On_EIT_Receive)( int iLength, BYTE* pBuffer );
```

- iLength is the length of the data buffer
- pBuffer points the the section table data

See ETSI 300 468 DVB documentation for further information of the EIT tables (Current/Next event, EPG data, etc). The function will be called any time a new table is received or received again.

Your plugin receive keyboard events of the main application window by exporting the following function:

Optional:

```
typedef int (*On_Key_Down)( UINT nChar, UINT nRepCnt, UINT nFlags );
```

- nChar is character code of the pressed key (for example: nChar == VK_RETURN when the return key is pressed)
- nRepCnt is the repeat count (see MSDN documentation for CWnd::OnKeyDown)
- nFlags (see MSDN documentation)

This function must return a value of type integer.

- Return 0 (zero) if you processed this event
- Return 1 if the key event shall NOT be sent to the main application window but other plugins may receive
- Return 2 if the key event shall be sent to the main application but NOT to other plugins
- Return 3 if this key shall be processed by your plugin only

Usually you should return always 0. You may use the other results on certain circumstances.

You may receive Remote Control events with the following function:

Optional:

```
typedef int (*On_RC5)( WORD wButtonCode, BOOL bButtonState );
```

- wButtonCode is the pressed button code of the remote control
- bButtonState is pressed (true) or released (false)

Currently this function is NOT SUPPORTED in WTV Ex 3.00!

You may receive the current application play state.

Optional:

```
typedef VOID (*On_Rec_Play)( int iState );
```

- iState is the current state of the application

The following states are defined:

```
typedef enum
{
    _MODE_TV,
    _MODE_PLAYBACK,
    _MODE_RECORD,
    _MODE_TIMESHIFT,
    _MODE_DVD,
    _MODE_RADIO,
    _MODE_PLAYBACK_INLINE
}PLAY_MODE;
```

The following function is called if a given PID filter will be closed.

Optional:

```
typedef VOID (*On_Filter_Close)( int iFilterId );
```

- iFilterId is identifier of the PID filter

The filter with the given id will stop receiving data shortly. This may happen when program will be exited.

You may receive the transport stream data of the current transponder.

(This refers to BDA devices only!)

Optional:

```
typedef VOID (*On_TransportStream_Receive)(BYTE* pBuffer); //188 byte packet
```

(added in WTVEx 3.01)

```
typedef VOID (*On_TransportStream_ReceiveBuffer)(BYTE* pBuffer, int iLength);
```

- pBuffer is a pointer to the first byte of the data buffer
- iLength is the length of the buffer (shall be a multiple of 188 bytes)

You will receive the whole TS data stream inside your plugin. Therefore you should return as fast as possible, because of the amount of data that will be passed here.

When you return not fast enough you may block the direct show BDA graph!

In place editing is possible but not recommended. The size of the buffer cannot be modified.

The first byte shall be 0x47, because you are receiving TS packets with headers here.

Sending commands to WTV.

Plugins may also send commands to WTV. A command can be sent via `SendMessage()` or `PostMessage()` by using the provided „hWnd“ in your „On_Start“ plugin function.

See MSDN Win32 documation for further information about `PostMessage()` or `SendMessage()`.

Just call „`SendMessage(hWnd, WM_USER, COMMAND_ID, IParameter)`“ to send a command to WTV.

The following commands are currently supported:

<code>#define WTVP_GET_PROGRAMM</code>	0x01020010
<code>#define WTVP_SET_PROGRAMM</code>	0x01020011
<code>#define WTVP_GET_PROGRAMM_NUMMER</code>	0x01020014
<code>#define WTVP_SET_PROGRAMM_NUMMER</code>	0x01020015
<code>#define WTVP_START_FILTER</code>	0x01020020
<code>#define WTVP_STOP_FILTER</code>	0x01020021
<code>#define WTVP_VOLUME_UP</code>	0x01020035
<code>#define WTVP_VOLUME_DOWN</code>	0x01020036
<code>#define WTVP_GET_VERSION</code>	0x01020100
<code>#define WTVP_CHANNELS_COUNT</code>	0x01021000
<code>#define WTVP_FAVORITES_COUNT</code>	0x01021002
<code>#define WTVP_GET_VOLUME</code>	0x01021010
<code>#define WTVP_SET_VOLUME</code>	0x01021011
<code>#define WTVP_SEND_DISEQC</code>	0x01021015
<code>#define WTVP_SET_CHANNEL_SID</code>	0x01021025
<code>#define WTVP_READ_REGISTRY</code>	0x01021046
<code>#define WTVP_WRITE_REGISTRY</code>	0x01021045
<code>#define WTVP_GET_TIMERRECORD</code>	0x01021051
<code>#define WTVP_SET_TIMERRECORD</code>	0x01021056
<code>#define WTVP_DEL_TIMERRECORD</code>	0x01021057
<code>#define WTVP_GET_TIMERCOUNT</code>	0x01021058
<code>#define WTVP_SET_OSD_SURFACE</code>	0x01021065
<code>#define WTVP_FILE_PLAYBACK</code>	0x01021070
<code>#define WTVP_OSD_SHOWMESSAGE</code>	0x01021085
<code>#define WTVP_RELOAD_CHANNELFILE</code>	0x01021095
<code>#define WTVP_SAVE_EPG</code>	0x01021098
<code>#define WTVP_GET_CURRENT_FILENAME</code>	0x01021110
<code>#define WTVP_SEND_DISEQC_EX</code>	0x01021120
<code>#define WTVP_LOAD_CHANNELFILE</code>	0x01021130
<code>#define WTVP_GET_CHANNELFILE_NAME</code>	0x01021140

You should only send those commands to WTV after your “On_Start” function has already been called (or inside “On_Start”).

You should never send those commands to WTV after your “On_Exit” function has been called and returned.

Command ID reference:

`#define WTVP_GET_PROGRAMM 0x01020010`

Send this command to retrieve a Tprogramm (see annex A) structure of the currently selected channel.

- IParam is a pointer to your TProgramm data structure. Data will be copied to this location.
-

`#define WTVP_SET_PROGRAMM 0x01020011`

Send this command to set the TProgramm (see annex A) data structure of the currently selected channel.

- IParam is a pointer to your TProgramm data structure.
-

`#define WTVP_GET_PROGRAMM_NUMMER 0x01020014`

Send this command to get the programm number of the currently selected channel.

- IParam is a pointer to your ChannelNumber data structure (see annex A).

RealNumber is the index of the currently viewed channel service.

VirtNumber is the index of a NVOD portal channel. Otherwise VirtNumber is equal to RealNumber.

`#define WTVP_SET_PROGRAMM_NUMMER 0x01020015`

Send this command to switch the channel by sending the desired channel number as IParam.

- IParam is a integer with the desired channel index
-

`#define WTVP_START_FILTER` `0x01020020`

- IParam is a pointer to a structure of type FilterInfo (see annex A)
- PluginIdentity member MUST be filled with your plugin's id number (passed by On_Start)
- FilterIdentity member MUST be filled. Just count your filter(s) from 0..23
- The FunctionPointer gives a pointer to your callback function and MUST be defined
- You MUST name the filter by filling out the "name" member
- Pid member defines the desired pid to filter

The provided function pointer must have the following prototype:

- `typedef VOID (*On_Filter_Data)(int iFilterID, int iLength, BYTE* pBuffer);`

Any time data is received this callback function will be called.

`#define WTVP_STOP_FILTER` `0x01020021`

- IParam is a 32 bit value
- lower 16 bit defines the plugin filter index to close
- upper 16 bit defines the plugin id that will close the filter (plugin_id was passed at the "On_Start" function)

example:

```
IParam = ( plugin_Id << 16 & 0xFFFF0000 ) | ( filter_id & 0xFFFF )
```

`#define WTVP_VOLUME_UP` `0x01020035`

Send this command to WTV to increase the current audio volume.

- IParam shall be NULL.
-

`#define WTVP_VOLUME_DOWN` `0x01020036`

Send this command to WTV to decrease the current audio volume.

- IParam shall be NULL.
-

`#define` WTVP_GET_VERSION 0x01020100

Send this command to WTV to get the version string of WTV. Describing program name and the version number.

- IParam points to an UCHAR string which receives the versioninfo string
-

`#define` WTVP_CHANNELS_COUNT 0x01021000

Send this command to WTV to get the number of channels inside the channel file.

- IParam points to an integer which receives the number of channels
-

`#define` WTVP_FAVORITES_COUNT 0x01021002

Send this command to WTV to get the number of favorites inside the channel file.

- IParam points to an integer which receives the number of favorites
-

`#define` WTVP_GET_VOLUME 0x01021010

Send this command to WTV to get the currently set volume.

- IParam points to a WORD which receives the current volume
-

`#define` WTVP_SET_VOLUME 0x01021011

Send this command to WTV to set the volume to the desired value (counting from 0 to 100). 0 = silence / 100 = max volume.

- IParam is the value of the volume to be set
-

`#define` WTVP_SEND_DISEQC 0x01021015

Send this command to WTV to send a diseqc message in case of a DVB-s device that support diseqc messages.

- IParam is a pointer to an array of 4 bytes in length (4 byte diseqc message)
-

`#define` WTVP_SET_CHANNEL_SID 0x01021025

Send this command to WTV to switch to a channel with the provided service id.

- IParam is the value of the service id of the desired channel
-

`#define` WTVP_READ_REGISTRY 0x01021046

Send this command to WTV to cause WTV to read its registry values.

- IParam shall be NULL.
-

`#define` WTVP_WRITE_REGISTRY 0x01021045

Send this command to WTV to cause WTV to write its registry values.

- IParam shall be NULL.
-

`#define` WTVP_GET_TIMERRECORD 0x01021051

Send this command to WTV to get a timer record entry.

- IParam points to a structure of type WTVP_TimerRecord (see annex A)
 - iTimerNumber must be filled to define the timer number you want
-

`#define WTVP_SET_TIMERRECORD` `0x01021056`

Send this command to WTV to set a timer record entry.

- IParam points to a structure of type `WTVP_TimerRecord` (see annex A)
 - The structure has to be filled with the new data
 - use `iTimerNumber` to edit an entry or to add a new one
-

`#define WTVP_DEL_TIMERRECORD` `0x01021057`

Send this command to WTV to delete a timer record entry.

- IParam value is the index of the timer entry to delete
-

`#define WTVP_GET_TIMERCOUNT` `0x01021058`

Send this command to WTV to get the number of timer scheduler entries.

- IParam points to an integer which receives upper bound index of timer entries
 - `length = upper bound + 1`
-

`#define WTVP_FILE_PLAYBACK` `0x01021070`

Send this command to WTV to open a media file for playback.

- IParam points to a character string which holds the full path and filename to the media file which shall be opened
-

`#define WTVP_OSD_SHOWMESSAGE` `0x01021085`

Send this command to WTV to show a message inside the OSD.

- IParam points to a character string which holds the string that will be shown in the OSD
-

`#define` WTVP_RELOAD_CHANNELFILE 0x01021095

Send this command to WTV to cause WTV to reload the currently loaded channel file.

- IParam shall be NULL
-

`#define` WTVP_SAVE_EPG 0x01021098

Send this command to WTV to cause WTV to save the epg cache file to the harddrive.

- IParam shall be NULL
-

`#define` WTVP_GET_CURRENT_FILENAME 0x01021110

Send this command to WTV to get the filename of the currently used file. (Media playback or a record filename).

- IParam points to a character string which receives the filename
-

`#define` WTVP_LOAD_CHANNELFILE 0x01021130

Send this command to WTV to cause WTV to load the channel file passed by the IParam Paramter.

- IParam points to a character string which receives the filename of the channel file to load
-

`#define` WTVP_GET_CHANNELFILE_NAME 0x01021140

Send this command to WTV to get the filename of the currently used channel file.

- IParam points to structure of type CHANNFILE_INFO (see annex A)
 - the pDestination member points to a CHAR buffer
 - the iSize member define to size of the buffer that will receive the channel file name
-

Plugin pull down menu

You can provide a pull down menu with your plugin which will be appended to the main applications plugins menu.

- Just add a menu resource to your plugin DLL
- use „EXTERN“ as menu resource name inside your DLL
- Command Ids of the menu must be in the range of 40000 – 49999
- Command Ids shall be unique
- Try not to use Ids which are already in used by other plugins

Hint:

You can get direct access to the menu with HMENU hMenu = ::GetMenu(hWnd), in case you wan't to check a menu item or something else.

Annex A (used data structures)

The following data structures are used.

```
typedef struct TProgramm
{
    CHAR        Name[30];
    CHAR        Provider[30];
    CHAR        Country[30];
    ULONG       frequency;
    UCHAR       Type;
    UCHAR       Voltage;
    UCHAR       AFC;
    UCHAR       DisEqC;
    UINT        SymbolRate;
    UCHAR       Quam;
    UCHAR       FEC;
    UCHAR       TVNorm;
    WORD        tp_id;
    WORD        Video_pid;
    WORD        Audio_pid;
    WORD        TeleText_pid;
    WORD        PMT_pid;
    WORD        PCR_pid;
    WORD        ECM_PID;
    WORD        SID_pid;
    WORD        AC3_pid;
    UCHAR       TVType;
    UCHAR       ServiceType;
    UCHAR       CA_ID;
    WORD        Temp_Audio;
    WORD        Filteranzahl;
}
```

```

    PIDFilters   Filters[32];
    WORD         CA_Anzahl;
    TCA_System   CA_System[32];
    CHAR         CA_Country[5];
    UCHAR        Merker;
    WORD         Link_TP;
    WORD         Link_SID;
    UCHAR        Dynamisch;
}TProgramm;

typedef struct
{
    unsigned short CA_Typ;
    unsigned short ECM;
    unsigned short EMM;
    unsigned int   Provider_Id;
} TCA_System;

typedef struct
{
    char           FilterName[5];
    unsigned char  FilterId;
    WORD           wPID;
}PIDFilters;

typedef struct
{
    CTime         Start; // 64bit value of the start time
    CTime         Stop; // 64bit value of the stop time
    char          szSendung[64];
    char          szSender[32];
    int           iChannelNumber;
    BOOL          bFav;
    int           iRecordingType;
    UINT          iRecordFormat;
    DWORD         dwFlags;
    DWORD         RESERVED2;
    DWORD         RESERVED3;
    DWORD         RESERVED4;
} TimerData2;

typedef struct
{
    int           iTimerNumber;
    int           iTimerCount;
    TimerData2   TimerData;
}WTVP_TimerRecord;

```

```
typedef struct
{
    char* pszDestination;
    int iSize;
} CHANNELFILE_INFO;
```

```
typedef struct
{
    int iRealNummer;
    int iVirtNummer;
}ChannelNumber;
```

```
typedef struct
{
    WORD wPluginIdentity;
    WORD wFilterIdentity;
    WORD wPid;
    UCHAR szName[32];
    DWORD dwFunctionPointer;
    INT iCurrentSign;
    INT iFilterType = 1 // STREAMING_FILTER;
}FilterInfo;
```