# SAWStudio

# FX APPLICATION PROGRAMMING INTERFACE

Version 1.1 - August 2001

Welcome to the **SAWStudio** FX API. This document provides all the necessary information and sample code to build FX modules that plug into the **SAWStudio** environment. These FX modules can process each track's audio data during *Real-Time* or non-*Real-Time* operations, can be patched *Pre* or *Post* fader on input and output tracks and can also be patched *Post* of the *Final Resolution* mix which includes all mixing and dithering options on the output tracks. The modules are allowed to freely withhold buffers or alter the buffer size during Real-Time operations and SAWStudio will maintain sync even though multiple modules are processing and returning varying sized buffers at the same time. Modules receive extremely accurate timing calls and are passed processing position and actual playback position information to facilitate accurate meter displays. In addition, modules may request Key Buffers from any of the 72 tracks at the same time as the streaming buffers to allow for Real-Time keying. The API also offers full parameter automation control in a simple and straightforward manner, and SAWStudio will track, save and maintain the automation data. The module may also control many of the SAWStudio features directly, such as playback positioning, area marking, solos, playback and record control, saving and restoring undo files, and much much more. The API is designed for the Windows 32 Bit operating systems and will run on Windows NT/2000.

The included test module code example provides a simple to use shell written in C that allows you to compile and see the inner workings of a simple FX module. You can also use this shell as a model for designing your own simple or complex plug-ins.

SAWStudio is designed as a large scale Real-Time Latency virtual mixing environment, which can replace large automated consoles and racks of processing equipment. If your module is performing Real-Time processing during playback, speed of buffer manipulation is critical. We suggest doing the actual buffer handling using assembly language to provide the highest speed possible, although in many cases optimized C code will work fine. Each FX module patched into a track takes time away from all tracks as it processes. Sluggish code will affect the overall track playback performance of the entire environment.

SAWStudio always runs in at least the 16 bit Hi Color mode which eliminates all concerns and needs for Windows palette handling. We encourage you to use 24 bit graphics bitmaps to display hi-res textured screens that look appealing as well as being functional.

There is no licensing fee of any kind required for the use of this API. Simply build a plug-in and distribute it via Shareware or produce it and market it as a third party add-on.

This FX protocol offers one of the most direct and powerful communications links to a high powered audio editing environment in the industry today, yet it is extremely easy to learn and control. We are excited for you to jump in and help enhance our editing environment while we strive to give your plug-ins one of the most enhanced platforms to showcase your work.

Feel free to call IQS at any time at 702-435-9077 and ask for assistance in getting your plug-ins up and running, or send email directly to BobL@iqsoft.com for support or more information.

Thanks for your interest.

## FX API AND TEST PLUG-IN SHELL OVERVIEW

The API is designed to communicate with standard 32 Bit Windows DLL Modules. The module must include each of the defined Function Procedures as exported functions. Even if a particular function is not needed by your module, you should include it as an empty function with just a RETURN statement.

The example code is written in C, although it is encased in a .cpp module. You can freely interact with C++ routines or inline assembly language if desired. You may use Visual C++, Borland C++, Symantec C++, or any Windows development platform that allows the creation of a standard 32 Bit Windows DLL.

The test module was written using the Microsoft Visual C++ version 6 environment and a project file is included. You should be able to open and compile the project directly if you are working in the same environment. To compile the test module in any other programming environment, create a new project targeted as a Windows 32 bit DLL and include all the files in the Test_PlugIn folder. The default calling convention of the project should be set to **__stdcall**.

There is a compiled debug .dll of the module included also for testing, if you can not compile your own.

The example code is a complete simple module that will compile into a working plug-in. In order to use the finished module, place the compiled DLL in the **SAWStudio_NativePlugIns** folder and edit the **SAWStudioFX_Native.ini** file to include, ***at the end***, the full name of the DLL, including the .dll extension. Make sure to include a carriage return at the end of the line.

Run **SAWStudio** and look for the plug-in in the FX Choices listbox.

Plug-ins can maintain multiple parameter sets of independent data for multiple patches. The API will keep track of which parameter set should be used for each independent call to each plug-in that is patched.

The test module sample code is very well commented and should be reviewed completely for a good understanding of the interaction and control of this API. Sample code segments are listed with each API function explanation and a complete detailed walk-through of the sample code is found at the end of this document.

## FX API SHARED MEMORY DATA STRUCTURE

This structure contains variables that are used for all communication between the **SAWStudio** environment and the FX Module. Very few of these variables are needed for actual simple buffer processing. The extra variables are used for in-depth control and interaction with the **SAWStudio** environment. The structure is defined as follows:

```
typedef struct
   {
   HWND              FX_hWndMain;
   HFONT             FX_IQSSysFont;
   DWORD             FX_VersionNum;

   DWORD             FX_Handler_DWord_Param1;
DWORD           FX_Handler_DWord_Param2;
DWORD           FX_Handler_DWord_Param3;
DWORD           FX_Handler_DWord_Param4;

int             FX_Handler_Int_Param1;
int             FX_Handler_Int_Param2;
int             FX_Handler_Int_Param3;
int             FX_Handler_Int_Param4;

LPBYTE          FX_Handler_Ptr_Param1;
LPBYTE          FX_Handler_Ptr_Param2;
LPBYTE          FX_Handler_Ptr_Param3;
LPBYTE          FX_Handler_Ptr_Param4;

DWORD           FX_Function_DWord_Param1;
DWORD           FX_Function_DWord_Param2;
DWORD           FX_Function_DWord_Param3;
DWORD           FX_Function_DWord_Param4;

int             FX_Function_Int_Param1;
int             FX_Function_Int_Param2;
int             FX_Function_Int_Param3;
int             FX_Function_Int_Param4;

LPBYTE          FX_Function_Ptr_Param1;
LPBYTE          FX_Function_Ptr_Param2;
LPBYTE          FX_Function_Ptr_Param3;
LPBYTE          FX_Function_Ptr_Param4;

DWORD           FX_MultiTrackRtnTrackOffset;
DWORD           FX_MultiTrackOutTrackOffset;
DWORD           FX_MultiTrackVideoTrackOffset;
DWORD           FX_MultiTrackControlTrackOffset;
DWORD           FX_MultiTrackLastTrackOffset;

HWND            FX_hWndMultiTrack;
DWORD           FX_MTSampleRate;
DWORD           FX_MTResolution;
DWORD           FX_MTZoomX;
DWORD           FX_MTZoomY;
DWORD           FX_MTMarkBegPos;
DWORD           FX_MTMarkEndPos;

char            FX_Title[48];
DWORD           FX_PlugInIndex;
DWORD           FX_ParamSet;
DWORD           FX_Track;
DWORD           FX_BufferRatioIn;
DWORD           FX_BufferRatioOut;

DWORD           FX_RequestOption;
DWORD           FX_RequestCancel;

DWORD           FX_ProcessFlag;
DWORD           FX_BufferByteSize;
DWORD           FX_BufferChans;
DWORD           FX_BufferBytesPerSample;
DWORD           FX_MaxBufferByteSize;

DWORD           FX_CurProcessPos;
DWORD           FX_CurSamplePos;
DWORD           FX_ProcessBegPos;
DWORD           FX_ProcessEndPos;

DWORD           FX_CurSmpteHours;
DWORD           FX_CurSmpteMins;
DWORD           FX_CurSmpteSecs;
DWORD           FX_CurSmpteFrames;
DWORD           FX_CurSmpteSFrames;
DWORD           FX_CurSmpteStartOffsetHours;
DWORD           FX_CurSmpteStartOffsetMins;
DWORD           FX_CurSmpteStartOffsetSecs;
DWORD           FX_CurSmpteStartOffsetFrames;
DWORD           FX_CurSmpteStartOffsetSFrames;
DWORD       FX_CurSmpteFrameCount;
DWORD       FX_CurSmpteFormat;
DWORD       FX_CurSmpteMode;

HWND        FX_hWndSoundFile;
DWORD       FX_SFSampleRate;
DWORD       FX_SFResolution;
DWORD       FX_SFZoomX;
DWORD       FX_SFZoomY;
int         FX_SFZoomShiftY;
DWORD       FX_SFZoomXMagnify;
DWORD       FX_SFMarkBegPos;
DWORD       FX_SFMarkEndPos;
DWORD       FX_SFCursorSamplePos;
DWORD       FX_SFStartSamplePos;

DWORD       FX_SFWaveDBScale;
int         FX_SFWaveCenterLft;
int         FX_SFWaveCenterRgt;
int         FX_SFWaveDivider;
int         FX_SFWaveTop;
int         FX_SFWaveBot;
int         FX_SFWaveLft;
int         FX_SFWaveRgt;
DWORD       FX_SFWaveOkToDraw;

DWORD       FX_MTWaveDBScale;
int         FX_MTWaveCenterLft;
int         FX_MTWaveCenterRgt;
int         FX_MTWaveDivider;
int         FX_MTWaveTop;
int         FX_MTWaveBot;
int         FX_MTWaveLft;
int         FX_MTWaveRgt;
int         FX_MTZoomShiftY;
DWORD       FX_MTWaveTrack;
DWORD       FX_MTCursorSamplePos;
```

```
    DWORD               FX_MTStartSamplePos;
    DWORD               FX_MTWaveOkToDraw;

    DWORD               FX_MTAutomationWriteFlag;
    DWORD               FX_XtraFlag2;
    DWORD               FX_XtraFlag3;
    DWORD               FX_XtraFlag4;
    DWORD               FX_XtraFlag5;
    DWORD               FX_XtraFlag6;
    DWORD               FX_XtraFlag7;
    DWORD               FX_XtraFlag8;
    DWORD               FX_XtraFlag9;
    DWORD               FX_XtraFlag10;
    DWORD               FX_XtraFlag11;
    DWORD               FX_XtraFlag12;

    DWORD               FX_CurTempoBeatsPerMin;
    DWORD               FX_CurTempoBeatsPerMeasure;
    DWORD               FX_CurTempoBeatRes;
    DWORD               FX_CurTempoTickRes;

    } IQS_SAWFXDATA;
```

A short description of each element follows:

**HWND FX_hWndMain;**

The **SAWStudio** Main window handle which you should use as a parent for displaying your module's windows and messages. This variable is always available.

**HFONT FX_IQSSysFont;**

The handle to the system font used by **SAWStudio**. This may be used in all Window's buttons and dialog boxes so that screen displays remain consistent in different screen resolutions. This variable is always available.

**DWORD FX_VersionNum;**

The current API version. This can be used as a reference for currently supported functions that might be needed by your plug-in. If the version number signals you that it is earlier than what your plug-in requires, you can set an internal flag to stop your plug-in from patching and display a warning message to the user at that time. This variable is always available.

**DWORD FX_Handler_DWord_Param1;**
**DWORD FX_Handler_DWord_Param2;**
**DWORD FX_Handler_DWord_Param3;**
**DWORD FX_Handler_DWord_Param4;**
**int FX_Handler_Int_Param1;**
**int FX_Handler_Int_Param2;**
**int FX_Handler_Int_Param3;**
**int FX_Handler_Int_Param4;**
**LPBYTE FX_Handler_Ptr_Param1;**
**LPBYTE FX_Handler_Ptr_Param2;**
**LPBYTE FX_Handler_Ptr_Param3;**
**LPBYTE FX_Handler_Ptr_Param4;**
Parameters of various types used by the Main Handler Routine which is always initiated by **SAWStudio** to communicate with the plug-in.

**DWORD FX_Function_DWord_Param1;**
**DWORD FX_Function_DWord_Param2;**
**DWORD FX_Function_DWord_Param3;**
**DWORD FX_Function_DWord_Param4;**
**int FX_Function_Int_Param1;**
**int FX_Function_Int_Param2;**
**int FX_Function_Int_Param3;**
**int FX_Function_Int_Param4;**
**LPBYTE FX_Function_Ptr_Param1;**
**LPBYTE FX_Function_Ptr_Param2;**
**LPBYTE FX_Function_Ptr_Param3;**
**LPBYTE FX_Function_Ptr_Param4;**
Parameters of various types used by the Execute Function Message which is always initiated by the plug-in to request something of **SAWStudio**.

**DWORD FX_MultiTrackRtnTrackOffset;**

The actual track offset value of the first *Return* track in **SAWStudio**. You may use this to determine what type of track your plug-in is patched into. This variable is always available.

**DWORD FX_MultiTrackOutTrackOffset;**
The actual track offset value of the first *Output* track in **SAWStudio**. You may use this to determine what type of track your plug-in is patched into. This variable is always available.

**DWORD FX_MultiTrackVideoTrackOffset;**
The actual track offset value of the *Video* track in **SAWStudio**. You may use this to determine what type of track your plug-in is patched into. This variable is always available.

**DWORD FX_MultiTrackControlTrackOffset;**
The actual track offset value of the *Control* track in **SAWStudio**. You may use this to determine what type of track your plug-in is patched into. This variable is always available.

**DWORD FX_MultiTrackLastTrackOffset;**
The actual track offset value of the last track in **SAWStudio**. You may use this to determine what type of track your plug-in is patched into. This variable is always available.

**HWND FX_hWndMultiTrack;**
The **SAWStudio** MultiTrack window handle. Use this to reference the MultiTrack window directly. This variable is always available.

**DWORD FX_MTSampleRate;**
The current session MultiTrack SampleRate. This variable is always available.

**DWORD FX_MTResolution;**
The current session MultiTrack Resolution. This variable is always available.

**DWORD FX_MTZoomX;**
The MultiTrack View horizontal zoom factor in samples per pixel. This variable is only updated with a call to the Execute Function REFRESH_FX_DATA (FX_WINDOW_DATA). A value of 1024 means that there are 1024 samples compressed into each pixel on the screen in the MultiTrack waveform display. This variable is useful if the plug-in is drawing reference data on top of the MultiTrack View waveform display.

**DWORD FX_MTZoomY;**

The MultiTrack View vertical scaling factor. This variable is only updated with a call to the Execute Function REFRESH_FX_DATA (FX_WINDOW_DATA). The actual data value is divided by this scaling factor to give the resulting pixel height in the MultiTrack waveform display. This variable is useful if the plug-in is drawing reference data on top of the MultiTrack View waveform display.

**DWORD FX_MTMarkBegPos;**

The MultiTrack View mark begin sample position. This variable is only updated with a call to the Execute Function REFRESH_FX_DATA (FX_WINDOW_DATA).

**DWORD FX_MTMarkEndPos;**

The MultiTrack View mark end sample position. This variable is only updated with a call to the Execute Function REFRESH_FX_DATA (FX_WINDOW_DATA).

**char FX_Title[48];**

Used to return a copy of your module's description to **SAWStudio** when your plug-in returns from the ***FxInit*** function call. This is the string that will appear in the FX Choices window in **SAWStudio**. The maximum length is 48 characters including the zero byte terminator at the end.

**DWORD FX_PlugInIndex;**

This value represents your plug-in's index position within **SAWStudio's** internal list of plug-in choices and is needed as a parameter for some of the requested execute function options. This variable is only accurate for your plug-in during the ***FxInit*** function call.

**DWORD FX_ParamSet;**

Parameter Set reference variable. Used by certain functions to pass parameter set values.

**DWORD FX_Track;**

Physical Track / Channel reference variable. This variable is zero based... Track 1 = 0, Track 2 = 1... etc. Used by certain functions to pass track values.

**DWORD FX_BufferRatioIn;**

Reserved for future use.

**DWORD FX_BufferRatioOut;**

Reserved for future use.

**DWORD FX_RequestOption;**

This variable is used by the plug-in to request options when returning from an FX_BEGIN_PROCESS call. Normally this variable is set to **0**. Setting this variable to one of the defined options will activate that request. The options are defined in the Header File as follows:

**#define FX_REQUEST_KEY_BUFFER 1**

**DWORD FX_RequestCancel;**

This variable is used by the plug-in to request a cancel when returning from an FX_BEGIN_PROCESS call. Normally this variable is set to **0**. Setting this variable to **1** causes the process operation to be canceled.

**DWORD FX_ProcessFlag;**

The current active process. This variable is always available. The options are defined in the Header File as follows:

**#define PROCESS_IDLE 0**
**#define REALTIME_PLAYBACK 1**
**#define BUILD_TO_MIX_FILE 2**
**#define PROCESS_TO_SOUND_FILE 3**
**#define BUILD_TO_FX_MODULES 4**
**#define DATA_PRESCAN 5**

**DWORD FX_BufferByteSize;**

The buffer byte size for the current *FxProcessBuffer* function call. This variable tells the plug-in how much data is available to process during the *FxProcessBuffer* function call. If the plug-in changes the buffer size, it must change this variable to reflect the new size for plug-ins which follow.

**DWORD FX_BufferChans;**

The number of data channels for the current *FxProcessBuffer* function call. This variable tells the plug-in how many channels the data is formatted to during the *FxProcessBuffer* function call.

**DWORD FX_BufferBytesPerSample;**

The number of bytes per sample per channel for the current *FxProcessBuffer* function call. This variable tells the plug-in how many bytes per sample the data is formatted to during the *FxProcessBuffer* function call. This value multiplied by the number of buffer channels dictates the actual number of bytes per sample in the buffer to be processed. This value is normally set to **2** for 16 bit resolution data and **4** for 20 and 24 bit resolution data. For a stereo buffer, each sample of 24 bit data would take up 8 bytes. (4 bytes per sample x 2 chans).

**DWORD FX_MaxBufferByteSize;**

This value represents the maximum buffer byte size that **SAWStudio** will allow for audio data streaming. Your plug-in must be prepared to handle a buffer of this size and allocate any extra internal processing buffers that it may need to meet this specification. Your plug-in must never return a buffer size larger than this value when processing during the ***FxProcessBuffer*** function call. This variable is always available.

### DWORD FX_CurProcessPos;

This value represents the actual process sample position of the first sample in the process buffer during the FxProcessBuffer function call. This is updated with each new buffer passed to the plug-ins for processing. This value will always lead the actual playback sample position because buffers are always processed ahead of time before they can be sent to the soundcard. The actual lead time depends on the latency settings and the dynamic latency routines that control how far in advance the buffers are processed.

### DWORD FX_CurSamplePos;

This value represents the actual current playback sample position. This value is updated with each call of the *FxChangePosition* function. This value can be used for *real-time* display of meters and other position related information.

### DWORD FX_ProcessBegPos;

The beginning process sample position. This value is set during the FX_BEGIN_PROCESS function call. The length of the upcoming process can be determined by the difference between the *Fx_ProcessEndPos* and *Fx_ProcessBegPos*.

### DWORD FX_ProcessEndPos;

The ending process sample position. This value is set during the FX_BEGIN_PROCESS function call. The length of the upcoming process can be determined by the difference between the *Fx_ProcessEndPos* and *Fx_ProcessBegPos*.

### DWORD FX_CurSmpteHours;

This value represents the actual current playback *Smpte Hours* position. This value is updated with each call of the *FxChangePosition* function. This value can be used for *real-time* display of meters and other position related information.

### DWORD FX_CurSmpteMins;

This value represents the actual current playback *Smpte Minutes* position. This value is updated with each call of the *FxChangePosition* function. This value can be used for *real-time* display of meters and other position related information.

### DWORD FX_CurSmpteSecs;

This value represents the actual current playback *Smpte Seconds* position. This value is updated with each call of the *FxChangePosition* function. This value can be used for *real-time* display of meters and other position related information.

### DWORD FX_CurSmpteFrames;

This value represents the actual current playback *Smpte Frames* position. This value is updated with each call of the *FxChangePosition* function. This value can be used for *real-time* display of meters and other position related information.

**DWORD FX_CurSmpteSFrames;**

This value represents the actual current playback *Smpte Sub-Frames* position. This value is updated with each call of the *FxChangePosition* function. This value can be used for *real-time* display of meters and other position related information.

**DWORD FX_CurSmpteStartOffsetHours;**

This value represents the actual current *Smpte Start Offset Hours* position. You can force an update of this value by using the Execute Function REFRESH_FX_DATA (FX_SMPTE_DATA)

**DWORD FX_CurSmpteStartOffsetMins;**

This value represents the actual current *Smpte Start Offset Minutes* position. You can force an update of this value by using the Execute Function REFRESH_FX_DATA (FX_SMPTE_DATA)

**DWORD FX_CurSmpteStartOffsetSecs;**

This value represents the actual current *Smpte Start Offset Seconds* position. You can force an update of this value by using the Execute Function REFRESH_FX_DATA (FX_SMPTE_DATA)

**DWORD FX_CurSmpteStartOffsetFrames;**

This value represents the actual current *Smpte Start Offset Frames* position. You can force an update of this value by using the Execute Function REFRESH_FX_DATA (FX_SMPTE_DATA)

**DWORD FX_CurSmpteStartOffsetSFrames;**

This value represents the actual current *Smpte Start Offset Sub-Frames* position. You can force an update of this value by using the Execute Function REFRESH_FX_DATA (FX_SMPTE_DATA)

**DWORD FX_CurSmpteFrameCount;**

This value represents the actual current playback *Smpte Total Frame Count* position. This value is updated with each call of the *FxChangePosition* function. This value can be used for *real-time* display of meters and other position related information.

**DWORD FX_CurSmpteFormat;**

This value represents the actual current *Smpte Format*. You can force an update of this value by using the Execute Function REFRESH_FX_DATA (FX_SMPTE_DATA)

The options are defined in the Header File as follows:

```
#define SMPTE_FORMAT_30N                1
#define SMPTE_FORMAT_30D                2
#define SMPTE_FORMAT_2997N          3
#define SMPTE_FORMAT_2997D          4
#define SMPTE_FORMAT_25                 5
#define SMPTE_FORMAT_24                 6
#define SMPTE_FORMAT_UNKNOWN            7
```

**DWORD FX_CurSmpteMode;**

This value represents the actual current *Smpte Mode*. You can force an update of this value by using the Execute Function REFRESH_FX_DATA (FX_SMPTE_DATA)

The options are defined in the Header File as follows:

```
#define SMPTE_NOT_ACTIVE                        0
#define SMPTE_GENERATE_ACTIVE                   1
#define SMPTE_GENERATE_MTC_ACTIVE               2
#define SMPTE_GENERATE_BOTH_ACTIVE              3
#define SMPTE_TRIGGER_ACTIVE                    4
#define SMPTE_TRIGGER_MTC_ACTIVE                5
#define SMPTE_CHASE_TRIGGER_ACTIVE              6
#define SMPTE_CHASE_TRIGGER_MTC_ACTIVE   7
```

**HWND FX_hWndSoundFile;**

The **SAWStudio** SoundFile window handle. Use this to reference the SoundFile window directly. This variable is always available.

**DWORD FX_SFSampleRate;**

The current active SoundFile SampleRate. This can be different from the actual session MultiTrack SampleRate value. The SoundFile view always plays back in the original SoundFile format. The MultiTrack can alter resolution and rates. You can force an update of this value by using the Execute Function REFRESH_FX_DATA (FX_WINDOW_DATA).

**DWORD FX_SFResolution;**

The current active SoundFile Resolution. This can be different from the actual session MultiTrack Resolution value. The SoundFile view always plays back in the original SoundFile format. The MultiTrack can alter resolution and rates. You can force an update of this value by using the Execute Function REFRESH_FX_DATA (FX_WINDOW_DATA).

**DWORD FX_SFZoomX;**

The SoundFile View horizontal zoom factor in samples per pixel. This variable is only updated with a call to the Execute Function REFRESH_FX_DATA (FX_WINDOW_DATA). A value of 1024 means that there are 1024 samples compressed into each pixel on the screen in the SoundFile waveform display. This variable is useful if the plug-in is drawing reference data on top of the SoundFile View waveform display.

**DWORD FX_SFZoomY;**

The SoundFile View vertical scaling factor. This variable is only updated with a call to the Execute Function REFRESH_FX_DATA (FX_WINDOW_DATA). The actual data value is divided by this scaling factor to give the resulting pixel height in the SoundFile waveform display. This variable is useful if the plug-in is drawing reference data on top of the SoundFile View waveform display.

**int FX_SFZoomShiftY;**

The SoundFile View vertical shift factor. This variable is only updated with a call to the Execute Function REFRESH_FX_DATA (FX_WINDOW_DATA). The actual scaled data value should be added to this plus/minus offset value to give the resulting pixel vertical position in the SoundFile waveform display. This variable is useful if the plug-in is drawing reference data on top of the SoundFile View waveform display.

**DWORD FX_SFZoomXMagnify;**

The SoundFile View horizontal Sample Edit Mode zoom factor in pixels per sample. This variable is only updated with a call to the Execute Function REFRESH_FX_DATA (FX_WINDOW_DATA). If Sample Edit Mode is not active, this value is zero. A value of 256 means that there are 256 pixels between samples. This variable is useful if the plug-in is drawing reference data on top of the SoundFile View waveform display.

**DWORD FX_SFMarkBegPos;**

The SoundFile View mark begin sample position. This variable is only updated with a call to the Execute Function REFRESH_FX_DATA (FX_WINDOW_DATA).

**DWORD FX_SFMarkEndPos;**

The SoundFile View mark end sample position. This variable is only updated with a call to the Execute Function REFRESH_FX_DATA (FX_WINDOW_DATA).

**DWORD FX_SFCursorSamplePos;**

The SoundFile View current cursor sample position. This variable is only updated with a call to the Execute Function REFRESH_FX_DATA (FX_WINDOW_DATA).

**DWORD FX_SFStartSamplePos;**

The SoundFile View current waveform left display edge sample position. This variable is only updated with a call to the Execute Function REFRESH_FX_DATA (FX_WINDOW_DATA).

**DWORD FX_SFWaveDBScale;**

The SoundFile View vertical dB waveform display height scaling factor. This variable is only updated with a call to the Execute Function REFRESH_FX_DATA (FX_WINDOW_DATA). The actual data value is divided by this scaling factor to give the resulting display area pixel height in the SoundFile waveform display. This variable is useful if the plug-in is drawing reference data on top of the SoundFile View waveform display.

**int FX_SFWaveCenterLft;**

The SoundFile View waveform display area left channel center line pixel value. This variable is only updated with a call to the Execute Function REFRESH_FX_DATA (FX_WINDOW_DATA). This variable is useful if the plug-in is drawing reference data on top of the SoundFile View waveform display.

**int FX_SFWaveCenterRgt;**

The SoundFile View waveform display area right channel center line pixel value. This variable is only updated with a call to the Execute Function REFRESH_FX_DATA (FX_WINDOW_DATA). This variable is useful if the plug-in is drawing reference data on top of the SoundFile View waveform display.

**int FX_SFWaveDivider;**

The SoundFile View waveform display area left / right channel divider line pixel value. This variable is only updated with a call to the Execute Function REFRESH_FX_DATA (FX_WINDOW_DATA). This variable is useful if the plug-in is drawing reference data on top of the SoundFile View waveform display.

**int FX_SFWaveTop;**

The SoundFile View waveform display area top rectangle pixel value. This variable is only updated with a call to the Execute Function REFRESH_FX_DATA (FX_WINDOW_DATA). This variable is useful if the plug-in is drawing reference data on top of the SoundFile View waveform display.

**int FX_SFWaveBot;**

The SoundFile View waveform display area bottom rectangle pixel value. This variable is only updated with a call to the Execute Function REFRESH_FX_DATA (FX_WINDOW_DATA). This variable is useful if the plug-in is drawing reference data on top of the SoundFile View waveform display.

**int FX_SFWaveLft;**

The SoundFile View waveform display area left rectangle pixel value. This variable is only updated with a call to the Execute Function REFRESH_FX_DATA (FX_WINDOW_DATA). This variable is useful if the plug-in is drawing reference data on top of the SoundFile View waveform display.

**int FX_SFWaveRgt;**

The SoundFile View waveform display area right rectangle pixel value. This variable is only updated with a call to the Execute Function REFRESH_FX_DATA (FX_WINDOW_DATA). This variable is useful if the plug-in is drawing reference data on top of the SoundFile View waveform display.

**DWORD FX_SFWaveOkToDraw;**

The SoundFile View OK to draw flag. This variable is only updated with a call to the Execute Function REFRESH_FX_DATA (FX_WINDOW_DATA). If this value is **0**, you must not draw to the SoundFile View window. If this value is **1**, it is ok to draw. This variable is useful if the plug-in is drawing reference data on top of the SoundFile View waveform display.

**DWORD FX_MTWaveDBScale;**

The MultiTrack View vertical dB waveform display height scaling factor. This variable is only updated with a call to the Execute Function REFRESH_FX_DATA (FX_WINDOW_DATA). The actual data value is divided by this scaling factor to give the resulting display area pixel height in the MultiTrack waveform display. This variable is useful if the plug-in is drawing reference data on top of the MultiTrack View waveform display.

**int FX_MTWaveCenterLft;**

The MultiTrack View waveform display area left channel center line pixel value. This variable is only updated with a call to the Execute Function REFRESH_FX_DATA (FX_WINDOW_DATA). This variable is useful if the plug-in is drawing reference data on top of the MultiTrack View waveform display.

**int FX_MTWaveCenterRgt;**

The MultiTrack View waveform display area right channel center line pixel value. This variable is only updated with a call to the Execute Function REFRESH_FX_DATA (FX_WINDOW_DATA). This variable is useful if the plug-in is drawing reference data on top of the MultiTrack View waveform display.

**int FX_MTWaveDivider;**

The MultiTrack View waveform display area left / right channel divider line pixel value. This variable is only updated with a call to the Execute Function REFRESH_FX_DATA (FX_WINDOW_DATA). This variable is useful if the plug-in is drawing reference data on top of the MultiTrack View waveform display.

**int FX_MTWaveTop;**

The MultiTrack View waveform display area top rectangle pixel value. This variable is only updated with a call to the Execute Function REFRESH_FX_DATA (FX_WINDOW_DATA). This variable is useful if the plug-in is drawing reference data on top of the MultiTrack View waveform display.

**int FX_MTWaveBot;**

The MultiTrack View waveform display area bottom rectangle pixel value. This variable is only updated with a call to the Execute Function REFRESH_FX_DATA (FX_WINDOW_DATA). This variable is useful if the plug-in is drawing reference data on top of the MultiTrack View waveform display.

**int FX_MTWaveLft;**

The MultiTrack View waveform display area left rectangle pixel value. This variable is only updated with a call to the Execute Function REFRESH_FX_DATA (FX_WINDOW_DATA). This variable is useful if the plug-in is drawing reference data on top of the MultiTrack View waveform display.

**int FX_MTWaveRgt;**

The MultiTrack View waveform display area right rectangle pixel value. This variable is only updated with a call to the Execute Function REFRESH_FX_DATA (FX_WINDOW_DATA). This variable is useful if the plug-in is drawing reference data on top of the MultiTrack View waveform display.

**int FX_MTZoomShiftY;**

The MultiTrack View vertical shift factor. This variable is only updated with a call to the Execute Function REFRESH_FX_DATA (FX_WINDOW_DATA). The actual scaled data value should be added to this plus/minus offset value to give the resulting pixel vertical position in the MultiTrack waveform display. This variable is useful if the plug-in is drawing reference data on top of the MultiTrack View waveform display.

**DWORD FX_MTWaveTrack;**

The MultiTrack View current track reference. This variable is only updated with a call to the Execute Function REFRESH_FX_DATA (FX_WINDOW_DATA).

**DWORD FX_MTCursorSamplePos;**

The MultiTrack View current cursor sample position. This variable is only updated with a call to the Execute Function REFRESH_FX_DATA (FX_WINDOW_DATA).

**DWORD FX_MTStartSamplePos;**

The MultiTrack View current waveform left display edge sample position. This variable is only updated with a call to the Execute Function REFRESH_FX_DATA (FX_WINDOW_DATA).

**DWORD FX_MTWaveOkToDraw;**

The MultiTrack View OK to draw flag. This variable is only updated with a call to the Execute Function REFRESH_FX_DATA (FX_WINDOW_DATA). If this value is **0**, you must not draw to the MultiTrack View window. If this value is **1**, it is ok to draw. This variable is useful if the plug-in is drawing reference data on top of the MultiTrack View waveform display.

**DWORD FX_MTAutomationWriteFlag;**

This variable reflects the status of Automation Write Mode. If this variable is **0**, automation writing is not active. If this variable is **1**, automation writing mode is active. This variable is always available.

**DWORD FX_XtraFlag2;**

Reserved for future use.

**DWORD FX_XtraFlag3;**
Reserved for future use.

**DWORD FX_XtraFlag4;**
Reserved for future use.

**DWORD FX_XtraFlag5;**
Reserved for future use.

**DWORD FX_XtraFlag6;**
Reserved for future use.

**DWORD FX_XtraFlag7;**
Reserved for future use.

**DWORD FX_XtraFlag8;**
Reserved for future use.

**DWORD FX_XtraFlag9;**
Reserved for future use.

**DWORD FX_XtraFlag10;**
Reserved for future use.

**DWORD FX_XtraFlag11;**
Reserved for future use.

**DWORD FX_XtraFlag12;**
Reserved for future use.

**DWORD FX_CurTempoBeatsPerMin;**

The MultiTrack View current Tempo Beats Per Minute value. You can force an update of this value by using the Execute Function REFRESH_FX_DATA (FX_TEMPO_DATA).

**DWORD FX_CurTempoBeatsPerMeasure;**

The MultiTrack View current Tempo Beats Per Measure value. You can force an update of this value by using the Execute Function REFRESH_FX_DATA (FX_TEMPO_DATA).

**DWORD FX_CurTempoBeatRes;**

The MultiTrack View current Tempo Beat Resolution value. You can force an update of this value by using the Execute Function REFRESH_FX_DATA (FX_TEMPO_DATA).

**DWORD FX_CurTempoTickRes;**

The MultiTrack View current Tempo Tick Resolution value. You can force an update of this value by using the Execute Function REFRESH_FX_DATA (FX_TEMPO_DATA).

## FX API FUNCTIONS

All communication between the plug-in and the host program is handled through the shared memory structure and four API functions. These four functions must be exported by the .dll module.

Small segments of extracted sample code will be used to help you see and understand how each of these functions are programmed.

### 1) FX Initialization Function

**void __stdcall FxInit (IQS_SAWFXDATA* FxPtrSawData)**

*IQS_SAWFXDATA* FxPtrSawData:* Pointer To Shared Memory Structure

This function is called once when **SAWStudio** first initializes itself and all FX plug-ins. This function passes a pointer to the shared memory data area allocated by **SAWStudio** which is defined as a structure in the module header file.

This function should store the *FxPtrSawData* pointer for future use, as this is the only time it will be passed.

The *FX_PlugInIndex* Shared Memory Variable should also be stored. This value represents your plug-in's index position within **SAWStudio's** internal list of plug-in choices and is needed as a parameter for some of the requested execute function options. This variable is only accurate for your plug-in at this time.

This function should also perform all necessary module initializations and set variable defaults so that the module is ready to be used.

If your plug-in allocates extra processing work buffers you should reference the *FX_MaxBufferByteSize* Shared Memory Variable. This value represents the maximum buffer byte size that **SAWStudio** will allow for audio data streaming. Your plug-in must be prepared to handle a buffer of this size.

Before returning from this function you should copy your module's description to the *FX_Title* Shared Memory Variable. This is the string that will appear in the FX Choices window in **SAWStudio**. The maximum length is 48 characters including the zero byte terminator at the end.

## Sample Code:

```
//============================
// Global Variables Available
// To All Module Routines
//============================

#define FX_ALL_PARAM_SETS               0xffffffff

char                        szFxTitle[] = "FX API TestPlugIn";
IQS_SAWFXDATA*              FxPtrSawData = NULL;

DWORD                       FxProgramVersion = 0;
DWORD                       FxPlugInIndex = 0;
DWORD                       FxMaxBufferSize = 0;
DWORD                       FxSampleRate;
DWORD                       FxResolution;
DWORD                       FxReturnTrkOffset;
DWORD                       FxOutputTrkOffset;

HWND                        hWndSawMain;
HFONT                       hIQSSysFont;

DWORD                       FxProcessFlag = 0;
DWORD                       FxPreScanFlag = 0;

DWORD                       FxProcessParamSet = 0;

//=======================
// SAWStudio API Routine
//
// FX Initialization
//=======================

void    __stdcall   FxInit(IQS_SAWFXDATA* SawFxDataPtr)

{
// Save Shared Memory Pointer
//----------------------------
FxPtrSawData = SawFxDataPtr;

// Take Data From Saw
//--------------------
FxPlugInIndex = FxPtrSawData->FX_PlugInIndex;
hWndSawMain = FxPtrSawData->FX_hWndMain;
hIQSSysFont = FxPtrSawData->FX_IQSSysFont;

FxProgramVersion = FxPtrSawData->FX_VersionNum;
FxMaxBufferSize = FxPtrSawData->FX_MaxBufferByteSize;

FxSampleRate = FxPtrSawData->FX_MTSampleRate;
FxResolution = FxPtrSawData->FX_MTResolution;

FxReturnTrkOffset = FxPtrSawData->FX_MultiTrackRtnTrackOffset;
FxOutputTrkOffset = FxPtrSawData->FX_MultiTrackOutTrackOffset;

// Initialize PlugIn
//------------------
InitializeProc();

// Reset All Parameter Sets
//-------------------------
ResetParamSet(FX_ALL_PARAM_SETS);

// Clear Process Variables
//------------------------
FxProcessParamSet = 0;
FxProcessFlag = PROCESS_IDLE;
FxPreScanFlag = PROCESS_IDLE;

// Send Back Title String
//-----------------------
lstrcpy(FxPtrSawData->FX_Title, szFxTitle);
return;
}
```

## 2) FX Main Handler Function

**void __stdcall FxMainHandler (DWORD FxFunctionID)**

*DWORD FxFunctionID:* Current Function ID value

This function is called whenever **SAWStudio** needs to initiate certain actions in the plug-in. The *FXFunctionID* parameter describes the particular function identifier for this call. The *FX_Handler* parameter Shared Memory Variables are used to pass parameters to and from the plug-in for these functions. This design allows the API to grow easily without affecting previously programmed modules.

## Sample Code:

```
//===========================
// SAWStudio Handler Routine
//
// FX Main Handler
//===========================

void  __stdcall  FxMainHandler(DWORD TempFunctionID)

{
// Call The Proper Routine Based On The Function ID
//-------------------------------------------------
if(TempFunctionID == FX_SHUTDOWN)
    {
    PlugInShutDown();
    return;
    }

if(TempFunctionID == FX_BEGIN_PROCESS)
    {
    PlugInBegProcess();
    return;
    }

if(TempFunctionID == FX_END_PROCESS)
    {
    PlugInEndProcess();
    return;
    }

// Etc
//----

return;
}
```

## Function ID Definitions:

### > FX_SHUTDOWN

This function is called when **SAWStudio** is about to be shutdown. Any cleanup required, such as destroying windows and freeing memory etc., should be done here.

### In Parameters Used:

None.

### Out Parameters Used:

None.

### Other Active Parameters Set By SAWStudio:
None.

### Sample Code:

```
//===========================
// SAWStudio Handler Routine
//
// PlugIn Shutdown
//===========================

void    __stdcall   PlugInShutDown(void)

{
// ShutDown PlugIn
//----------------
ShutdownProc();

return;
}
```

### > FX_BEGIN_PROCESS
This function is called at the beginning of any process that uses the FX Modules.

### In Parameters Used:

| | |
|---|---|
| *FX_Handler_DWord_Param1:* | Parameter Set |
| *FX_Handler_DWord_Param2:* | Bytes Per Sample |
| *FX_Handler_DWord_Param3:* | Number Of Channels |
| *FX_Handler_DWord_Param4:* | Buffer Byte Size |

### Out Parameters Used:

| | |
|---|---|
| *FX_RequestOption:* | Request Special Option |
| *FX_RequestCancel:* | Request To Cancel Process |

### Other Active Parameters Set By SAWStudio:
*FX_ProcessFlag:* Tells the function which type of process is happening on this call. The options are defined in the Header File as follows:

**#define PROCESS_IDLE 0**
**#define REALTIME_PLAYBACK 1**
**#define BUILD_TO_MIX_FILE 2**

```
#define PROCESS_TO_SOUND_FILE          3
#define BUILD_TO_FX_MODULES            4
#define DATA_PRESCAN                   5
```

If this call defines a *Real-Time Playback* process and your module does not perform in *real-time*, you should either ignore this call or warn the user with a message.

*FX_MTSampleRate:* Set to current process samplerate. This could be different from the session samplerate on BuildMix processes. If variables need to be adjusted accordingly, do so now.

*FX_MTResolution:* Set to current process resolution. This could be different from the session resolution on *BuildMix* processes. If variables need to be adjusted accordingly, do so now.

*FX_MaxBufferByteSize:* Set to the maximum buffer size allowed for the process operation. Your plug-in might be asked to process this much data, and your plug-in must not process and send back more than this much data.

*FX_ProcessBegPos:* Set to the starting process position.

*FX_ProcessEndPos:* Set to the ending process position.

*FX_CurSmpteFormat:* Set to the current Smpte Format. These options are defined in the Header File as follows:

```
#define SMPTE_FORMAT_30N                         1
#define SMPTE_FORMAT_30D                         2
#define SMPTE_FORMAT_2997N             3
#define SMPTE_FORMAT_2997D             4
#define SMPTE_FORMAT_25                          5
#define SMPTE_FORMAT_24                          6
#define SMPTE_FORMAT_UNKNOWN                     7
```

*FX_CurSmpteMode:* Set to the current Smpte Mode. These options are defined in the Header File as follows:

```
#define SMPTE_NOT_ACTIVE                         0
#define SMPTE_GENERATE_ACTIVE                    1
#define SMPTE_GENERATE_MTC_ACTIVE                2
#define SMPTE_GENERATE_BOTH_ACTIVE               3
#define SMPTE_TRIGGER_ACTIVE                     4
```

**#define SMPTE_TRIGGER_MTC_ACTIVE 5**
**#define SMPTE_CHASE_TRIGGER_ACTIVE 6**
**#define SMPTE_CHASE_TRIGGER_MTC_ACTIVE 7**

If a module allows itself to be patched in multiple locations at the same time, it must have different variable parameter sets that define all settings for each patch assignment. The FX_Handler_DWord_Param1 variable tells the module which parameter set to work with on this call. All variables that would constitute a complete parameter set should be defined as arrays indexed from zero up to the maximum number of parameter sets your module allows.

Since the module may be patched into the 24 bit or the 16 bit data path, the FX_Handler_DWord_Param2 and FX_Handler_DWord_Param3 Shared Memory Variables provide the module with details about the buffer resolution which is to be used and the number of channels in the buffer. This information may be needed for the module to properly initialize its data variables for processing. The FX_Handler_DWord_Param2 (Bytes Per Sample) value will be set to 2 for 16 bit resolution buffers and 4 for 20 and 24 bit resolution buffers. The data alignment for the 20 and 24 bit data will be set at the 24 bit position of the DWORD, sign extended upwards and zero extended downwards. Multiplying the Bytes Per Sample and the Number Of Channels values together tells your module the number of actual bytes per sample in the data buffers to be processed.

The FX_Handler_DWord_Param4 Shared Memory Variable provides the starting buffer size in bytes. Plug-ins are allowed to alter the buffer size at will, so each new buffer must be checked for the current valid data size. You must maintain the buffer byte size to an exact sample boundary, according to the actual bytes per sample value calculated above.

This function should initialize any variables it needs just prior to buffer processing.

Your plug-in may request special options by placing an Option code in the FX_RequestOption Shared Memory Variable before returning from this function. The codes are defined in the Header File as follows:

**#define FX_REQUEST_KEY_BUFFER 1**
Your plug-in may request key data from any of the SAWStudio Input Tracks by using this code.

Place the track number (starting at zero) of interest into the FX_Handler_Int_Param1.

Place a pointer to an internally allocated key buffer of size FX_MaxBufferByteSize in the FX_Handler_Ptr_Param1. SAWStudio will then copy the appropriate synced key buffer data into your specified key buffer each time the FXProcessBuffer function passes you new data.

Your plug-in may request to cancel the current process by setting the FX_RequestCancel Shared Memory Variable to 1 before returning from this function.

## Sample Code:

```
//============================
// Global Variables Available
// To All Module Routines
//============================

#define FX_ALL_CONTROLS                   0xffffffff
#define DATA_PRESCAN                      5
```

```
IQS_SAWFXDATA*              FxPtrSawData;

DWORD                       FxProcessFlag = 0;
DWORD                       FxPreScanFlag = 0;

DWORD                       FxProcessParamSet = 0;
DWORD                       FxProcessBytesPerSample = 4;

//=========================
// SAWStudio Handler Routine
//
// PlugIn Begin Process
//=========================

void    __stdcall   PlugInBegProcess(void)

{
// Set Process Variables For This Call
//------------------------------------
FxProcessParamSet = FxPtrSawData->FX_Handler_DWord_Param1;
FxProcessBytesPerSample = FxPtrSawData->FX_Handler_DWord_Param2;
FxProcessFlag = FxPtrSawData->FX_ProcessFlag;

if(FxProcessFlag == DATA_PRESCAN)
    FxPreScanFlag = 1;
else
    FxPreScanFlag = 0;

// Init This Parameter Set For Processing
//---------------------------------------
InitPlugInVariables(FxProcessParamSet, FX_ALL_CONTROLS);
InitPlugInMeter(FxProcessParamSet);

return;
}
```

### > FX_END_PROCESS

This function is called at the end of any process that uses the FX Modules.

**In Parameters Used:**

*FX_Handler_DWord_Param1:* Parameter Set

**Out Parameters Used:**

None.

**Other Active Parameters Set By SAWStudio:**

*FX_ProcessFlag:* Set to **PROCESS_IDLE**.

The *FX_Handler_DWord_Param1* variable is the current parameter set that should be worked with on this call.

This function should cleanup any variables that it needs to clear at the completion of data processing.

**Sample Code:**

```
//============================
// Global Variables Available
// To All Module Routines
//============================

#define PROCESS_IDLE                    0

IQS_SAWFXDATA*              FxPtrSawData;

DWORD                       FxProcessFlag = 0;
DWORD                       FxPreScanFlag = 0;

DWORD                       FxProcessParamSet = 0;

//===========================
// SAWStudio Handler Routine
//
// PlugIn End Process
//===========================

void    __stdcall   PlugInEndProcess(void)

{
// Reset Process Variables
//------------------------
FxProcessParamSet = FxPtrSawData->FX_Handler_DWord_Param1;

FxProcessFlag = PROCESS_IDLE;
FxPreScanFlag = PROCESS_IDLE;

// CleanUp Parameter Set After Processing If Needed
//------------------------------------------------
ResetMeterDisplay(FxProcessParamSet);

return;
}
```

### > FX_BEGIN_XTRA

Reserved for future use.

### > FX_END_XTRA

Reserved for future use.

### > FX_OPEN_PLUGIN_WINDOW

This function is called when **SAWStudio** requests the plug-in window to be displayed, or the user double-clicks on the plug-in name in one of the Patch window listboxes.

#### In Parameters Used:

| | |
|---|---|
| *FX_Handler_DWord_Param1:* | Parameter Set |
| *FX_Handler_DWord_Param2:* | Track (zero based) |
| *FX_Handler_DWord_Param3:* | Handle to Patch Window |
| *FX_Handler_DWord_Param4:* | Bytes Per Sample |

#### Out Parameters Used:

None.

#### Other Active Parameters Set By SAWStudio:

None.

This function should open up the module's setup window or dialog box and allow the user to have access to all control parameters that are required to operate the module. The window should use the *FX_hWndMain* as the parent window.

If a module allows itself to be patched in multiple locations at the same time, it must have different variable parameter sets that define all settings for each patch assignment. The *FX_Handler_DWord_Param1* variable tells the module which parameter set to work with on this call. All variables that would constitute a complete parameter set should be defined as arrays indexed from zero up to the maximum number of parameter sets your module allows.

The *FX_Handler_DWord_Param2* variable contains the current track that the plug-in patch is referencing.

The *FX_Handler_DWord_Param3* variable contains the window handle of the patch window that is being clicked. This window could be used as one reference for positioning your module window if desired. This window should not be used as a parent for your module window, because the patch window can be in a hidden or visible state based on F-Key Workspace assignments.

If the module does not have adjustable parameters then the user should be told with a message that there is *No Setup Required*.

#### Sample Code:

```
//============================
// Global Variables Available
// To All Module Routines
//============================

IQS_SAWFXDATA*              FxPtrSawData;

//===========================
// SAWStudio Handler Routine
//
// PlugIn Open Window
//===========================

void    __stdcall   PlugInOpenWindow(void)

{
// Temp Variables
//---------------
DWORD   TempParamSet;
HWND    TempRefWindow;

// Get Parameter Data
//-------------------
TempParamSet = FxPtrSawData->FX_Handler_DWord_Param1;
TempRefWindow = (HWND)FxPtrSawData->FX_Handler_DWord_Param3;

// Open PlugIn Window
//-------------------
CreatePlugInWindow(TempParamSet, TempRefWindow);
return;
}
```

### > FX_CLOSE_PLUGIN_WINDOW

This function is called when **SAWStudio** requests the plug-in window to be hidden.

### In Parameters Used:

*FX_Handler_DWord_Param1:* Parameter Set

### Out Parameters Used:

None.

### Other Active Parameters Set By SAWStudio:

None.

Each module's window and dialog box should have a close button so the user can manually remove the display. The module should also respond to this function call so that **SAWStudio** can remove the display when it needs to.

### Sample Code:

```
//============================
// Global Variables Available
// To All Module Routines
//============================
IQS_SAWFXDATA*              FxPtrSawData;

//============================
// SAWStudio Handler Routine
//
// PlugIn Close Window
//============================

void    __stdcall   PlugInCloseWindow(void)

{
// Temp Variables
//---------------
DWORD   TempParamSet;

// Get Parameter Data
//-------------------
TempParamSet = FxPtrSawData->FX_Handler_DWord_Param1;

// Close PlugIn Window
//--------------------
DestroyPlugInWindow(TempParamSet);
return;
}
```

## > FX_PATCH_NEW_PARAM_SET

This function is called each time a plug-in is patched into any of the FX Patch Windows on any console channel.

### In Parameters Used:

| | |
|---|---|
| *FX_Handler_DWord_Param1:* | Bytes Per Sample |
| *FX_Handler_DWord_Param2:* | Track (zero based) |

### Out Parameters Used:

| | |
|---|---|
| *FX_ParamSet:* | New ParamSet Value |

### Other Active Parameters Set By SAWStudio:

None.

If a module allows itself to be patched in multiple locations at the same time, it must have different variable parameter sets that define all settings for each patch assignment. All variables that would constitute a complete parameter set should be defined as arrays indexed from zero up to the maximum number of parameter sets your module allows.

Before returning from this function, the plug-in should set the *FX_ParamSet* variable to the next available parameter set number. **SAWStudio** will use this value every time it communicates with this instance of the plug-in. If there are no more available parameter sets then the plug-in should set the *FX_ParamSet* variable to the defined NONE_AVAILABLE constant (0xFFFFFFFF).

Modules that use more than one parameter set should keep track of the assigned sets in a *ModuleParamSetTable* array. Each element in the array should be set to 1 or cleared to 0 as that set is assigned or reset. The assignment and clearing of module parameter sets can occur in a random order and must be kept track of by the module.

The *FX_Handler_DWord_Param1* variable provides the module with details about the buffer resolution which is to be used for this parameter set. This information may be needed for the module to properly preset its initial parameters.

The *FX_Handler_DWord_Param2* variable provides the module with details about the track that the module is patched into.

## Sample Code:

```
//=============================
// Global Variables Available
// To All Module Routines
//=============================

#define MAX_PARAM_SETS                  16
#define NONE_AVAILABLE                  0xffffffff

IQS_SAWFXDATA*              FxPtrSawData;

DWORD                       PlugInBytesPerSample[MAX_PARAM_SETS];
DWORD                       PlugInTrack[MAX_PARAM_SETS];

DWORD                       PlugInParamSetCount;
DWORD                       PlugInParamSetTable[MAX_PARAM_SETS];

char                        WarningMsg[] = "WARNING!";

char                        MaximumAssignmentMsg[] =
                            "This Effect Has Reached Its Maximum Assignments!";

//=================================
// SAWStudio Handler Routine
//
// PlugIn Patch New Parameter Set
//=================================

void    __stdcall   PlugInPatchNewParamSet(void)

{
// Temp Variables
//---------------
DWORD   TempParamSet;

// Are There Any Available Parameter Sets?
//-----------------------------------------
if(PlugInParamSetCount < MAX_PARAM_SETS)
    {
    // Search For The First Available Set
    //------------------------------------
    for(TempParamSet = 0; TempParamSet < MAX_PARAM_SETS; TempParamSet++)
        {
        if(PlugInParamSetTable[TempParamSet] == 0)
            {
            // Save Format Information
            //-------------------------
            PlugInBytesPerSample[TempParamSet] = FxPtrSawData->FX_Handler_DWord_Param1;

            // Save Track Information
            //------------------------
            PlugInTrack[TempParamSet] = FxPtrSawData->FX_Handler_DWord_Param2;

            // Init PlugIn Variables If Needed
            //--------------------------------
            InitPlugInVariables(TempParamSet, FX_ALL_CONTROLS);

            // Pass New Parameter Set Value Back To Saw
            //-----------------------------------------
            FxPtrSawData->FX_ParamSet = TempParamSet;

            // Mark This Set Active In Table
            //-----------------------------
            PlugInParamSetTable[TempParamSet] = 1;

            // Increment Active Set Count
            //--------------------------
            PlugInParamSetCount++;
            return;
            }
        }
    }

// None Available
//---------------
MessageBox(NULL, MaximumAssignmentMsg, WarningMsg, MB_TASKMODAL | MB_TOPMOST | MB_ICONEXCLAMATION | MB_OK);

FxPtrSawData->FX_ParamSet = NONE_AVAILABLE;
return;
}
```

## > FX_RESET_PARAM_SET

This function is called each time a module is removed from any of the FX Patch Windows on any console channel.

### In Parameters Used:

*FX_Handler_DWord_Param1:* Parameter Set

### Out Parameters Used:

None.

## Other Active Parameters Set By SAWStudio:

None.

If a module allows itself to be patched in multiple locations at the same time, it must have different variable parameter sets that define all settings for each patch assignment. The *FX_Handler_DWord_Param1* variable tells the module which parameter set to work with on this call. All variables that would constitute a complete parameter set should be defined as arrays indexed from zero up to the maximum number of parameter sets your module allows.

As parameter sets are cleared the ***ModuleParamSetTable*** array should be adjusted so that this parameter set can now be made available to be assigned to other tracks.

The plug-in display window should be removed if visible and all variables cleared for this parameter set.

## Sample Code:

```
//============================
// Global Variables Available
// To All Module Routines
//============================

#define MAX_PARAM_SETS                  16

IQS_SAWFXDATA*              FxPtrSawData;

HWND                        hWndPlugIn[MAX_PARAM_SETS];

//============================
// SAWStudio Handler Routine
//
// PlugIn Reset Parameter Set
//============================

void    __stdcall   PlugInResetParamSet(void)

{
// Temp Variables
//---------------
DWORD   TempParamSet;

// Get Parameter Data
//-------------------
TempParamSet = FxPtrSawData->FX_Handler_DWord_Param1;

// Reset Parameter Set
//--------------------
ResetParamSet(TempParamSet);

// If The Window Exists... Close It
//---------------------------------
if(hWndPlugIn[TempParamSet])
    DestroyPlugInWindow(TempParamSet);
return;
}
```

## > FX_RECEIVE_PARAM_SET_DATA

This function is called whenever a new EDL is opened, or a plug-in is copied from one console channel to another. Plug-in data which has been saved or copied is sent back to the plug-in with this function call.

### In Parameters Used:

| | |
|---|---|
| *FX_Handler_DWord_Param1:* | Parameter Set |
| *FX_Handler_DWord_Param2:* | Buffer Byte Size |
| *FX_Handler_Ptr_Param1:* | Pointer To Buffer Containing The Data |

### Out Parameters Used:

| | |
|---|---|
| *FX_Handler_DWord_Param3:* | Return 0 If Data Is Received Correctly<br>Return 1 If Data Is Not Correct Format |

### Other Active Parameters Set By SAWStudio:

None.

The *FX_Handler_DWord_Param1* variable contains the parameter set to work with on this call. It may be a single parameter set value or the define value **FX_ALL_PARAM_SETS**.

The *FX_Handler_Ptr_Param1* variable points to a memory buffer containing the parameter data. The *FX_Handler_DWord_Param2* variable contains the size of the buffer data in bytes.

The module should check the first 32 Bytes for a header ID String identifying the data format. If the data format is correct, it should load the data from the buffer into its parameter storage area. The plug-in should set the *FX_Handler_DWord_Param3* variable to **0** if the data is received correctly.

If there is a problem with the data format, the plug-in should set the *FX_Handler_DWord_Param3* variable to **1** to signal **SAWStudio** that there was corrupted plug-in data.

### Sample Code:

```
//============================
// Global Variables Available
// To All Module Routines
//============================

#define MAX_PARAM_SETS                  16
#define MAX_AUTO_CONTROLS               8
#define FX_ALL_PARAM_SETS               0xffffffff
```

```
IQS_SAWFXDATA*              FxPtrSawData;

DWORD                       PlugInBytesPerSample[MAX_PARAM_SETS];
DWORD                       PlugInTrack[MAX_PARAM_SETS];

DWORD                       PlugInParamSetCount;
DWORD                       PlugInParamSetTable[MAX_PARAM_SETS];

int                         PlugInXStart[MAX_PARAM_SETS];
int                         PlugInYStart[MAX_PARAM_SETS];
int                         PlugInWidth[MAX_PARAM_SETS];
int                         PlugInHeight[MAX_PARAM_SETS];

int                         PlugInDataValue[MAX_PARAM_SETS][MAX_AUTO_CONTROLS];
int                         PlugInDefaultDataValue[MAX_PARAM_SETS][MAX_AUTO_CONTROLS];

//===================================
// SAWStudio Handler Routine
//
// PlugIn Receive Parameter Set Data
//===================================

void    __stdcall   PlugInReceiveParamSetData(void)

{
// Temp Variables
//---------------
DWORD   TempParamSet;
DWORD   TempBuffSize;
LPBYTE  TempBuffPtr;
DWORD   TempOffset;
DWORD   TempCount;
DWORD   TempMaxCount;
DWORD   TempSet;
DWORD   TempVisibleFlag;

// Get Parameter Data
//-------------------
TempParamSet = FxPtrSawData->FX_Handler_DWord_Param1;
TempBuffSize = FxPtrSawData->FX_Handler_DWord_Param2;
TempBuffPtr = FxPtrSawData->FX_Handler_Ptr_Param1;

// Check Header For A Match
//-------------------------
if(lstrcmp((LPSTR)TempBuffPtr, "FX API TEST PLUGIN PRESETS 001 ") != 0)
    {
    // Not Proper Format
    //------------------
    FxPtrSawData->FX_Handler_DWord_Param3 = 1;
    return;
    }

// Offset Past The Header
//-----------------------
```

```
TempOffset = 32;

// Reset The Parameter Set(s) (Might Be All Sets)
//-----------------------------------------------
ResetParamSet(TempParamSet);

// Set Max Count
//--------------
if(TempParamSet == FX_ALL_PARAM_SETS)
    TempMaxCount = MAX_PARAM_SETS;
else
    TempMaxCount = 1;

// Read Individual Settings For Each Active ParamSet
//--------------------------------------------------
for(TempCount = 0; TempCount <= TempMaxCount; TempCount++)
    {
    // Get Parameter Set Value
    //------------------------
    TempSet = *(PDWORD)(TempBuffPtr + TempOffset);
    TempOffset = TempOffset + 4;

    // End Of List... Exit Loop
    //-------------------------
    if(TempSet == END_OF_LIST)
        break;

    // Read PlugIn Data For This Set
    //------------------------------
    PlugInTrack[TempSet] = *(PDWORD)(TempBuffPtr + TempOffset);
    TempOffset = TempOffset + 4;

    PlugInBytesPerSample[TempSet] = *(PDWORD)(TempBuffPtr + TempOffset);
    TempOffset = TempOffset + 4;

    PlugInDefaultDataValue[TempSet][AUTO_BYPASS_SW] = *(PDWORD)(TempBuffPtr + TempOffset);
    TempOffset = TempOffset + 4;

    PlugInDefaultDataValue[TempSet][AUTO_VOL_POT] = *(PDWORD)(TempBuffPtr + TempOffset);
    TempOffset = TempOffset + 4;

    PlugInDefaultDataValue[TempSet][AUTO_VALUE_1] = *(PDWORD)(TempBuffPtr + TempOffset);
    TempOffset = TempOffset + 4;

    PlugInDefaultDataValue[TempSet][AUTO_VALUE_2] = *(PDWORD)(TempBuffPtr + TempOffset);
    TempOffset = TempOffset + 4;

    PlugInDefaultDataValue[TempSet][AUTO_VALUE_3] = *(PDWORD)(TempBuffPtr + TempOffset);
    TempOffset = TempOffset + 4;

    PlugInDefaultDataValue[TempSet][AUTO_VALUE_4] = *(PDWORD)(TempBuffPtr + TempOffset);
    TempOffset = TempOffset + 4;

    PlugInDefaultDataValue[TempSet][AUTO_VALUE_5] = *(PDWORD)(TempBuffPtr + TempOffset);
    TempOffset = TempOffset + 4;

    PlugInDefaultDataValue[TempSet][AUTO_VALUE_6] = *(PDWORD)(TempBuffPtr + TempOffset);
    TempOffset = TempOffset + 4;

    // Set All Current Values To New Defaults
    //---------------------------------------
    CopyMemory(&PlugInDataValue[TempSet][0], &PlugInDefaultDataValue[TempSet][0], MAX_AUTO_CONTROLS * 4);

    // Init PlugIn Variables
    //----------------------
    InitPlugInVariables(TempSet, FX_ALL_CONTROLS);

    // Mark This Set Active In Table
    //------------------------------
    PlugInParamSetTable[TempSet] = 1;

    // Increment Active Set Count
    //---------------------------
    PlugInParamSetCount++;

    // Read Window Info
    //-----------------
    PlugInXStart[TempSet] = *(PDWORD)(TempBuffPtr + TempOffset);
    TempOffset = TempOffset + 4;

    PlugInYStart[TempSet] = *(PDWORD)(TempBuffPtr + TempOffset);
    TempOffset = TempOffset + 4;

    PlugInWidth[TempSet] = *(PDWORD)(TempBuffPtr + TempOffset);
    TempOffset = TempOffset + 4;

    PlugInHeight[TempSet] = *(PDWORD)(TempBuffPtr + TempOffset);
    TempOffset = TempOffset + 4;

    TempVisibleFlag = *(PDWORD)(TempBuffPtr + TempOffset);
    TempOffset = TempOffset + 4;

    // Display Window If Needed
    //-------------------------
    if(TempVisibleFlag)
        CreatePlugInWindow(TempSet, NULL);
    }

// Everything OK
//--------------
FxPtrSawData->FX_Handler_DWord_Param3 = 0;

return;
}
```

**> FX_SEND_PARAM_SET_DATA**

This function is called whenever an EDL is saved, or a plug-in is copied from one console channel to another. Plug-in data is requested from the plug-in with this function call.

### In Parameters Used:

| | |
|---|---|
| *FX_Handler_DWord_Param1:* | Parameter Set |
| *FX_Handler_Ptr_Param1:* | Pointer To Buffer Which Receives The Data |

### Out Parameters Used:

| | |
|---|---|
| *FX_Handler_DWord_Param2:* | Byte Size Of Buffer Sent Back |

### Other Active Parameters Set By SAWStudio:

None.

The *FX_Handler_DWord_Param1* variable contains the parameter set to work with on this call. It may be a single parameter set value or the define value **FX_ALL_PARAM_SETS**.

The *FX_Handler_Ptr_Param1* variable points to a memory buffer ready to receive the parameter data. The maximum buffer size is 256K Bytes. The module should use the first 32 Bytes for a zero terminated header ID String identifying the data as its own, with format version information (31 bytes plus the zero). The module should copy its parameter set data into the buffer and return the number of bytes to **SAWStudio** using the *FX_Handler_DWord_Param2* variable.

If the module needs greater than the 256K Bytes, it should store a reference filename in the buffer and use an external file to store the data.

If the module has not been patched and no parameter sets are active at the time of this call, the module should return zero byte size using the *FX_Handler_DWord_Param2* variable.

Data storage space required for your parameter sets should be minimized so that the EDL file size does not grow unnecessarily large.

### Sample Code:

```
//==============================
// Global Variables Available
// To All Module Routines
//==============================

#define MAX_PARAM_SETS                  16
#define MAX_AUTO_CONTROLS               8
#define FX_ALL_PARAM_SETS               0xffffffff

IQS_SAWFXDATA*              FxPtrSawData;

DWORD                       PlugInBytesPerSample[MAX_PARAM_SETS];
DWORD                       PlugInTrack[MAX_PARAM_SETS];

DWORD                       PlugInParamSetCount;
DWORD                       PlugInParamSetTable[MAX_PARAM_SETS];
```

```
int                          PlugInXStart[MAX_PARAM_SETS];
int                          PlugInYStart[MAX_PARAM_SETS];
int                          PlugInWidth[MAX_PARAM_SETS];
int                          PlugInHeight[MAX_PARAM_SETS];

int                          PlugInDataValue[MAX_PARAM_SETS][MAX_AUTO_CONTROLS];
int                          PlugInDefaultDataValue[MAX_PARAM_SETS][MAX_AUTO_CONTROLS];

//================================
// SAWStudio Handler Routine
//
// PlugIn Send Parameter Set Data
//================================

void    __stdcall   PlugInSendParamSetData(void)

{
// Temp Variables
//---------------
DWORD   TempParamSet;
DWORD   TempBuffSize;
LPBYTE  TempBuffPtr;
DWORD   TempSet;
DWORD   TempMaxCount;
RECT    rWindow;

// Get Parameter Data
//-------------------
TempParamSet = FxPtrSawData->FX_Handler_DWord_Param1;
TempBuffPtr = FxPtrSawData->FX_Handler_Ptr_Param1;

// If No Active Parameter Sets... Return Zero Buffer Size
//-------------------------------------------------------
if(PlugInParamSetCount == 0)
    {
    FxPtrSawData->FX_Handler_DWord_Param2 = 0;
    return;
    }

// Set Max Count (Might Be All Sets)
//----------------------------------
if(TempParamSet == FX_ALL_PARAM_SETS)
    TempMaxCount = MAX_PARAM_SETS;
else
    TempMaxCount = 1;

lstrcpy((LPSTR)TempBuffPtr, "FX API TEST PLUGIN PRESETS 001 ");
TempBuffSize = 32;

// Save Individual Settings For Each Active ParamSet
//--------------------------------------------------
for(TempSet = 0; TempSet < TempMaxCount; TempSet++)
    {
```

```
// Skip If Not Active
//-------------------
if(PlugInParamSetTable[TempSet] == 0)
    continue;

// Write Parameter Set Value
//--------------------------
*(PDWORD)(TempBuffPtr + TempBuffSize) = TempSet;
TempBuffSize = TempBuffSize + 4;

// Write PlugIn Data
//------------------
*(PDWORD)(TempBuffPtr + TempBuffSize) = PlugInTrack[TempSet];
TempBuffSize = TempBuffSize + 4;

*(PDWORD)(TempBuffPtr + TempBuffSize) = PlugInBytesPerSample[TempSet];
TempBuffSize = TempBuffSize + 4;

*(PDWORD)(TempBuffPtr + TempBuffSize) = PlugInDefaultDataValue[TempSet][AUTO_BYPASS_SW];
TempBuffSize = TempBuffSize + 4;

*(PDWORD)(TempBuffPtr + TempBuffSize) = PlugInDefaultDataValue[TempSet][AUTO_VOL_POT];
TempBuffSize = TempBuffSize + 4;

*(PDWORD)(TempBuffPtr + TempBuffSize) = PlugInDefaultDataValue[TempSet][AUTO_VALUE_1];
TempBuffSize = TempBuffSize + 4;

*(PDWORD)(TempBuffPtr + TempBuffSize) = PlugInDefaultDataValue[TempSet][AUTO_VALUE_2];
TempBuffSize = TempBuffSize + 4;

*(PDWORD)(TempBuffPtr + TempBuffSize) = PlugInDefaultDataValue[TempSet][AUTO_VALUE_3];
TempBuffSize = TempBuffSize + 4;

*(PDWORD)(TempBuffPtr + TempBuffSize) = PlugInDefaultDataValue[TempSet][AUTO_VALUE_4];
TempBuffSize = TempBuffSize + 4;

*(PDWORD)(TempBuffPtr + TempBuffSize) = PlugInDefaultDataValue[TempSet][AUTO_VALUE_5];
TempBuffSize = TempBuffSize + 4;

*(PDWORD)(TempBuffPtr + TempBuffSize) = PlugInDefaultDataValue[TempSet][AUTO_VALUE_6];
TempBuffSize = TempBuffSize + 4;

// Write Window Info
//------------------
if((hWndPlugIn[TempSet]) == 0)
    {
    // Not Visible
    //------------
    *(PDWORD)(TempBuffPtr + TempBuffSize) = 0;
    TempBuffSize = TempBuffSize + 4;

    *(PDWORD)(TempBuffPtr + TempBuffSize) = 0;
    TempBuffSize = TempBuffSize + 4;
```

```
        *(PDWORD)(TempBuffPtr + TempBuffSize) = 0;
        TempBuffSize = TempBuffSize + 4;

        *(PDWORD)(TempBuffPtr + TempBuffSize) = 0;
        TempBuffSize = TempBuffSize + 4;

        *(PDWORD)(TempBuffPtr + TempBuffSize) = 0;
        TempBuffSize = TempBuffSize + 4;
        }
    else
        {
        // Visible
        //--------
        GetWindowRect(hWndPlugIn[TempSet], &rWindow);

        *(PDWORD)(TempBuffPtr + TempBuffSize) = rWindow.left;
        TempBuffSize = TempBuffSize + 4;

        *(PDWORD)(TempBuffPtr + TempBuffSize) = rWindow.top;
        TempBuffSize = TempBuffSize + 4;

        *(PDWORD)(TempBuffPtr + TempBuffSize) = rWindow.right - rWindow.left;
        TempBuffSize = TempBuffSize + 4;

        *(PDWORD)(TempBuffPtr + TempBuffSize) = rWindow.bottom - rWindow.top;
        TempBuffSize = TempBuffSize + 4;

        *(PDWORD)(TempBuffPtr + TempBuffSize) = 1;
        TempBuffSize = TempBuffSize + 4;
        }
    }

// Write End Of List
//------------------
*(PDWORD)(TempBuffPtr + TempBuffSize) = END_OF_LIST;
TempBuffSize = TempBuffSize + 4;

// Return Buffer Size
//-------------------
FxPtrSawData->FX_Handler_DWord_Param2 = TempBuffSize;

return;
}
```

## > FX_INIT_PARAM_SET_DATA

This function is called whenever **SAWStudio** requires the plug-in to initialize all controls of an individual parameter set for some special reason.

### In Parameters Used:

*FX_Handler_DWord_Param1:* Parameter Set

### Out Parameters Used:

None.

### Other Active Parameters Set By SAWStudio:

None.

The *FX_Handler_DWord_Param1* variable contains the parameter set to work with on this call.

### Sample Code:

```
//============================
// Global Variables Available
// To All Module Routines
//============================

#define FX_ALL_CONTROLS                  0xffffffff

IQS_SAWFXDATA*              FxPtrSawData;

//================================
// SAWStudio Handler Routine
//
// PlugIn Init Parameter Set Data
//================================

void    __stdcall   PlugInInitParamSetData(void)

{
// Temp Variables
//---------------
DWORD   TempParamSet;

// Get Parameter Data
//-------------------
TempParamSet = FxPtrSawData->FX_Handler_DWord_Param1;

// Init PlugIn Variables
//---------------------
InitPlugInVariables(TempParamSet, FX_ALL_CONTROLS);

return;
}
```

## > FX_CHANGE_MT_RATE

This function is called for all currently patched plug-ins whenever the MultiTrack samplerate is changed.

### In Parameters Used:

*FX_Handler_DWord_Param1:* New SampleRate

## Out Parameters Used:

None.

## Other Active Parameters Set By SAWStudio:

None.

The FX_Handler_DWord_Param1 variable contains the new samplerate. The module should perform any necessary variable adjustments to the new samplerate for all parameter sets at this time.

## Sample Code:

```
//============================
// Global Variables Available
// To All Module Routines
//============================

#define MAX_PARAM_SETS                  16
#define FX_ALL_CONTROLS                 0xffffffff

IQS_SAWFXDATA*              FxPtrSawData;

DWORD                       FxSampleRate;
DWORD                       PlugInParamSetTable[MAX_PARAM_SETS];

//===========================
// SAWStudio Handler Routine
//
// PlugIn Change Sample Rate
//===========================

void    __stdcall   PlugInChangeRate(void)

{
// Temp Variables
//---------------
DWORD   TempParamSet;
DWORD   TempNewRate;

// Get Parameter Data
//-------------------
TempNewRate = FxPtrSawData->FX_Handler_DWord_Param1;

// Init New Data If Rate Is Different
//-----------------------------------
if(FxSampleRate != TempNewRate)
    {
    FxSampleRate = TempNewRate;
    for(TempParamSet = 0; TempParamSet < MAX_PARAM_SETS; TempParamSet++)
        {
        if(PlugInParamSetTable[TempParamSet] == 0)
            continue;

        // Init PlugIn Variables
        //-----------------------
        InitPlugInVariables(TempParamSet, FX_ALL_CONTROLS);
        }
    }

return;
}
```

## > FX_CHANGE_MT_RES

This function is called for all currently patched plug-ins whenever the MultiTrack bit resolution is changed.

### In Parameters Used:

*FX_Handler_DWord_Param1:* New Resolution

### Out Parameters Used:

None.

### Other Active Parameters Set By SAWStudio:

None.

The FX_Handler_DWord_Param1 variable contains the new resolution. The module should perform any necessary variable adjustments to the new resolution for all parameter sets at this time.

### Sample Code:

```
//============================
// Global Variables Available
// To All Module Routines
//============================

#define MAX_PARAM_SETS                   16
#define FX_ALL_CONTROLS                  0xffffffff

IQS_SAWFXDATA*              FxPtrSawData;

DWORD                       FxResolution;
DWORD                       PlugInParamSetTable[MAX_PARAM_SETS];

//===========================
// SAWStudio Handler Routine
//
// PlugIn Change Resolution
//==========================

void    __stdcall   PlugInChangeRes(void)

{
// Temp Variables
//---------------
DWORD   TempParamSet;
DWORD   TempNewRes;

// Get Parameter Data
//-------------------
TempNewRes = FxPtrSawData->FX_Handler_DWord_Param1;

// Init New Data If Res Is Different
//----------------------------------
if(FxResolution != TempNewRes)
    {
    FxResolution = TempNewRes;

    for(TempParamSet = 0; TempParamSet < MAX_PARAM_SETS; TempParamSet++)
        {
        if(PlugInParamSetTable[TempParamSet] == 0)
            continue;

        // Init PlugIn Variables
        //----------------------
        InitPlugInVariables(TempParamSet, FX_ALL_CONTROLS);
        }
    }

return;
}
```

### > FX_CHANGE_MT_ACTIVE_TRACK

This function is called for all currently patched plug-ins whenever the MultiTrack active track is changed.

**In Parameters Used:**

*FX_Handler_DWord_Param1:* New Track (zero based)

**Out Parameters Used:**

None.

**Other Active Parameters Set By SAWStudio:**

None.

The *FX_Handler_DWord_Param1* variable contains the new active track.  The module may use this information to update displays or change parameters on a single display to reflect the proper track settings.

### Sample Code:

```
//============================
// Global Variables Available
// To All Module Routines
//============================

IQS_SAWFXDATA*              FxPtrSawData;

//============================
// SAWStudio Handler Routine
//
// PlugIn Change Active Track
//============================

void    __stdcall   PlugInChangeActiveTrack(void)

{
// Temp Variables
//---------------
DWORD   TempNewTrack;

// Get Parameter Data
//-------------------
TempNewTrack = FxPtrSawData->FX_Handler_DWord_Param1;

// Perform Any Track Change Related Process Needed
//------------------------------------------------

return;
}
```

### > FX_CHANGE_MT_MARKS

This function is called for all currently patched plug-ins whenever the MultiTrack mark begin or end position is changed.

### In Parameters Used:

| | |
|---|---|
| *FX_Handler_DWord_Param1:* | New MT Mark Begin Sample Position |
| *FX_Handler_DWord_Param2:* | New MT Mark End Sample Position |

### Out Parameters Used:

None.

### Other Active Parameters Set By SAWStudio:

None.

The *FX_Handler_DWord_Param1* and *FX_Handler_DWord_Param2* variables contain the new mark begin and end sample positions. The module may use this information to update displays or change parameters on a single display to reflect the proper settings.

### Sample Code:

```
//==============================
// Global Variables Available
// To All Module Routines
//==============================

IQS_SAWFXDATA*              FxPtrSawData;

//=============================
// SAWStudio Handler Routine
//
// PlugIn Change MT Marks
//=============================

void    __stdcall   PlugInChangeMTMarks(void)

{
// Temp Variables
//----------------
DWORD   TempNewMarkBegPos;
DWORD   TempNewMarkEndPos;

// Get Parameter Data
//--------------------
TempNewMarkBegPos = FxPtrSawData->FX_Handler_DWord_Param1;
TempNewMarkEndPos = FxPtrSawData->FX_Handler_DWord_Param2;

// Perform Any Marked Area Change Related Process Needed
//-------------------------------------------------------

return;
}
```

## > FX_CHANGE_SMPTE_INFO

This function is called for all patched plug-ins whenever the Smpte Start Offset, Smpte Mode or Smpte Format is changed.

### In Parameters Used:

None.

### Out Parameters Used:

None.

### Other Active Parameters Set By SAWStudio:

None.

The *FX_CurSmpteStartOffsetSFrames* Shared Memory Variable contains the Smpte Start Offset sub frame value.

The *FX_CurSmpteStartOffsetFrames* Shared Memory Variable contains the Smpte Start Offset frame value.

The *FX_CurSmpteStartOffsetSecs* Shared Memory Variable contains the Smpte Start Offset seconds value.

The *FX_CurSmpteStartOffsetMins* Shared Memory Variable contains the Smpte Start Offset minutes value.

The *FX_CurSmpteStartOffsetHours* Shared Memory Variable contains the Smpte Start Offset Hours value.

The *FX_CurSmpteFormat* Shared Memory Variable contains the Smpte Format value.

These options are defined in the Header File as follows:

```
#define SMPTE_FORMAT_30N                1
#define SMPTE_FORMAT_30D                2
#define SMPTE_FORMAT_2997N          3
#define SMPTE_FORMAT_2997D          4
#define SMPTE_FORMAT_25                 5
#define SMPTE_FORMAT_24                 6
#define SMPTE_FORMAT_UNKNOWN            7
```

The *FX_CurSmpteMode* Shared Memory Variable contains the Smpte Mode value.

These options are defined in the Header File as follows:

```
#define SMPTE_NOT_ACTIVE                        0
#define SMPTE_GENERATE_ACTIVE                   1
#define SMPTE_GENERATE_MTC_ACTIVE               2
#define SMPTE_GENERATE_BOTH_ACTIVE              3
#define SMPTE_TRIGGER_ACTIVE                    4
#define SMPTE_TRIGGER_MTC_ACTIVE                5
#define SMPTE_CHASE_TRIGGER_ACTIVE              6
#define SMPTE_CHASE_TRIGGER_MTC_ACTIVE   7
```

### Sample Code:

```
//===========================
// Global Variables Available
// To All Module Routines
//===========================

IQS_SAWFXDATA*              FxPtrSawData;

//==========================
// SAWStudio Handler Routine
//
// PlugIn Change Smpte Info
//==========================

void    __stdcall   PlugInChangeSmpteInfo(void)

{
// The Smpte Offset Shared Memory Variables Are Set
// The Smpte Format Shared Memory Variable Is Set
// The Smpte Mode Shared Memory Variable Is Set
//
// Perform Any Smpte Info Change Related Process Needed
//----------------------------------------------------

return;
}
```

## > FX_CHANGE_TEMPO_INFO

This function is called for all patched plug-ins whenever any of the Tempo variables are changed.

### In Parameters Used:

None.

### Out Parameters Used:

None.

### Other Active Parameters Set By SAWStudio:

None.

The *FX_CurTempoBeatsPerMin* Shared Memory Variable contains the Tempo beats per minute value. This value is multiplied by 1000 to maintain three decimal places.

The *FX_CurTempoBeatsPerMeasure* Shared Memory Variable contains the Tempo beats per measure value. This value represents the numerator of the time signature.

The *FX_CurTempoBeatRes* Shared Memory Variable contains the Tempo beat resolution value. This value represents the denominator of the time signature.

The *FX_CurTempoTickRes* Shared Memory Variable contains the Tempo tick resolution value. This value is multiplied by 10 to maintain 1 decimal place.

**Sample Code:**

```
//=============================
// Global Variables Available
// To All Module Routines
//=============================

IQS_SAWFXDATA*              FxPtrSawData;

//============================
// SAWStudio Handler Routine
//
// PlugIn Change Tempo Info
//============================

void    __stdcall   PlugInChangeTempoInfo(void)

{
// The Tempo Shared Memory Variables Are Set
//
// Perform Any Tempo Info Change Related Process Needed
//----------------------------------------------------

return;
}
```

## > FX_CLEAR_CHASE_AUTOMATION

This function is called for all currently patched plug-ins whenever the MultiTrack cursor position is changed during idle play mode. Any track that contains automation data will first send this message to force all plug-ins to clear their controls to their default values to start an automation chase sequence to the current position.

The module should first copy all current control values to duplicate storage locations for later comparison. Then, the module should immediately update all control values to their default values. Other module variables do not need to be initialized at this time.

If your module does not handle automation, simply ignore this call.

**In Parameters Used:**

| | |
|---|---|
| *FX_Handler_DWord_Param1:* | Parameter Set |
| *FX_Handler_DWord_Param2:* | Track (zero based) |

**Out Parameters Used:**

None.

**Other Active Parameters Set By SAWStudio:**

None.

## Sample Code:

```
//============================
// Global Variables Available
// To All Module Routines
//============================

#define MAX_PARAM_SETS                  16
#define MAX_AUTO_CONTROLS               8

IQS_SAWFXDATA*              FxPtrSawData;

DWORD                       PlugInTrack[MAX_PARAM_SETS];
DWORD                       PlugInParamSetCount;
DWORD                       PlugInParamSetTable[MAX_PARAM_SETS];

int                         PlugInDataValue[MAX_PARAM_SETS][MAX_AUTO_CONTROLS];
int                         PlugInDefaultDataValue[MAX_PARAM_SETS][MAX_AUTO_CONTROLS];
int                         PlugInAutoDataValue[MAX_PARAM_SETS][MAX_AUTO_CONTROLS];

//===========================
// SAWStudio Handler Routine
//
// Clear Chase Automation
//===========================

void    __stdcall   PlugInClearChaseAutomation(void)

{
// Temp Variables
//---------------
DWORD   TempParamSet;
DWORD   TempPhysicalTrack;
DWORD   TempSet;
DWORD   TempStartSet;
DWORD   TempEndSet;

// No Active Parameter Sets... Nothing To Do
//------------------------------------------
if(PlugInParamSetCount == 0)
    return;

// Get Parameter Data
//-------------------
TempParamSet = FxPtrSawData->FX_Handler_DWord_Param1;
TempPhysicalTrack = FxPtrSawData->FX_Handler_DWord_Param2;

// Set Start And End Loop Values (Might Be All Sets)
//--------------------------------------------------
if(TempParamSet == FX_ALL_PARAM_SETS)
```

```
    {
    TempStartSet = 0;
    TempEndSet = MAX_PARAM_SETS;
    }
else
    {
    // Verify A Track Match For This Parameter Set
    //-------------------------------------------
    if(TempPhysicalTrack != PlugInTrack[TempParamSet])
        return;

    TempStartSet = TempParamSet;
    TempEndSet = TempParamSet + 1;
    }

for(TempSet = TempStartSet; TempSet < TempEndSet; TempSet++)
    {
    // Skip All Inactive Parameter Sets
    //--------------------------------
    if(PlugInParamSetTable[TempSet] == 0)
        continue;

    // Copy Current Data Values To Automation Data Storage
    //---------------------------------------------------
    CopyMemory(&PlugInAutoDataValue[TempSet][0], &PlugInDataValue[TempSet][0], MAX_AUTO_CONTROLS * 4);

    // Set Current Data Values To Default Values
    //-----------------------------------------
    CopyMemory(&PlugInDataValue[TempSet][0], PlugInDefaultDataValue[TempSet][0], MAX_AUTO_CONTROLS * 4);
    }

return;
}
```

## > FX_STORE_CHASE_AUTOMATION

This function is called for all currently patched plug-ins whenever the MultiTrack cursor position is changed during idle play mode. Any track that contains automation data sends this message to each designated plug-in to allow the plug-in to update its current data value for each control that has been automated up to the current cursor position.

The module should immediately update the control value to its new value. The updates should not be initialized or displayed at this time, as any single control might receive multiple update messages as the automation database is scanned from the start position to the current cursor position.

If your module does not handle automation, simply ignore this call.

### In Parameters Used:

| | |
|---|---|
| *FX_Handler_DWord_Param1:* | Parameter Set |
| *FX_Handler_DWord_Param2:* | Track (zero based) |
| *FX_Handler_DWord_Param3:* | Control ID# |

*FX_Handler_int_Param1:* New Control Data Value

## Out Parameters Used:

None.

## Other Active Parameters Set By SAWStudio:

None.

## Sample Code:

```
//==============================
// Global Variables Available
// To All Module Routines
//==============================

#define MAX_PARAM_SETS                  16
#define MAX_AUTO_CONTROLS               8

IQS_SAWFXDATA*              FxPtrSawData;

DWORD                       PlugInTrack[MAX_PARAM_SETS];
DWORD                       PlugInParamSetTable[MAX_PARAM_SETS];

int                         PlugInDataValue[MAX_PARAM_SETS][MAX_AUTO_CONTROLS];

//============================
// SAWStudio Handler Routine
//
// Store Chase Automation
//============================

void    __stdcall   PlugInStoreChaseAutomation(void)

{
// Temp Variables
//---------------
DWORD   TempParamSet;
DWORD   TempPhysicalTrack;
DWORD   TempCtrlId;
int     TempDataValue;

// Get Parameter Data
//-------------------
TempParamSet = FxPtrSawData->FX_Handler_DWord_Param1;

// Skip If Inactive
//-----------------
if(PlugInParamSetTable[TempParamSet] == 0)
    return;
TempPhysicalTrack = FxPtrSawData->FX_Handler_DWord_Param2;

// Verify A Track Match For This Parameter Set
//-----------------------------------------
if(TempPhysicalTrack != PlugInTrack[TempParamSet])
    return;

// Set New Data Value
//-------------------
TempCtrlId = FxPtrSawData->FX_Handler_DWord_Param3;
TempDataValue = FxPtrSawData->FX_Handler_Int_Param1;

PlugInDataValue[TempParamSet][TempCtrlId] = TempDataValue;
return;
}
```

## > FX_DISPLAY_CHASE_AUTOMATION

This function is called for all currently patched plug-ins whenever the MultiTrack cursor position is changed during idle play mode. Any track that contains automation data will send this message after the automation database is scanned to force all plug-ins to display any updated or changed controls at the end of the automation chase sequence to the current position.

The module should compare all current control values to the duplicate storage locations saved earlier. Then, the module should immediately update the display for each control value which has changed during the chase scan sequence.

If your module does not handle automation, simply ignore this call.

### In Parameters Used:

| | |
|---|---|
| *FX_Handler_DWord_Param1:* | Parameter Set |
| *FX_Handler_DWord_Param2:* | Track (zero based) |

### Out Parameters Used:

None.

### Other Active Parameters Set By SAWStudio:

None.

### Sample Code:

```
//============================
// Global Variables Available
// To All Module Routines
//============================

#define MAX_PARAM_SETS                  16
```

```
#define MAX_AUTO_CONTROLS               8

IQS_SAWFXDATA*              FxPtrSawData;

DWORD                       PlugInTrack[MAX_PARAM_SETS];
DWORD                       PlugInParamSetCount;
DWORD                       PlugInParamSetTable[MAX_PARAM_SETS];
HWND                        hWndPlugIn[MAX_PARAM_SETS];
HDC                         PlugInDC[MAX_PARAM_SETS];

int                         PlugInDataValue[MAX_PARAM_SETS][MAX_AUTO_CONTROLS];
int                         PlugInAutoDataValue[MAX_PARAM_SETS][MAX_AUTO_CONTROLS];

//===========================
// SAWStudio Handler Routine
//
// Display Chase Automation
//===========================

void    __stdcall   PlugInDisplayChaseAutomation(void)

{
// Temp Variables
//---------------
DWORD   TempParamSet;
DWORD   TempPhysicalTrack;
DWORD   TempSet;
DWORD   TempStartSet;
DWORD   TempEndSet;

// No Active Parameter Sets... Nothing To Display
//-----------------------------------------------
if(PlugInParamSetCount == 0)
    return;

// Get Parameter Data
//-------------------
TempParamSet = FxPtrSawData->FX_Handler_DWord_Param1;
TempPhysicalTrack = FxPtrSawData->FX_Handler_DWord_Param2;

// Set Start And End Loop Values (Might Be All Sets)
//--------------------------------------------------
if(TempParamSet == FX_ALL_PARAM_SETS)
    {
    TempStartSet = 0;
    TempEndSet = MAX_PARAM_SETS;
    }
else
    {
    // Verify A Track Match For This Parameter Set
    //-------------------------------------------
    if(TempPhysicalTrack != PlugInTrack[TempParamSet])
        return;
    TempStartSet = TempParamSet;
    TempEndSet = TempParamSet + 1;
    }

for(TempSet = TempStartSet; TempSet < TempEndSet; TempSet++)
    {
    // Skip If Window Is Not Open
    //--------------------------
    if(hWndPlugIn[TempSet] == 0)
        continue;

    // Display All Changed Values
    //--------------------------
    if(PlugInDataValue[TempSet][AUTO_BYPASS_SW] != PlugInAutoDataValue[TempSet][AUTO_BYPASS_SW])
        {
        if(PlugInDataValue[TempSet][AUTO_BYPASS_SW])
            DisplayBitmapBtn(PlugInDC[TempSet], &RectPlugInBypassDisplayBtn, hPlugInBypassBtnBitmap);
        else
            RefreshBitmap(PlugInDC[TempSet], &RectPlugInBypassDisplayBtn, hPlugInMainBitmap);
        }

    if(PlugInDataValue[TempSet][AUTO_VOL_POT] != PlugInAutoDataValue[TempSet][AUTO_VOL_POT])
        DrawVertKnob(PlugInDC[TempSet], &RectPlugInVolPot, PlugInDataValue[TempSet][AUTO_VOL_POT], 1);

    if(PlugInDataValue[TempSet][AUTO_VALUE_1] != PlugInAutoDataValue[TempSet][AUTO_VALUE_1])
        DisplayPlugInValue(PlugInDC[TempSet], AUTO_VALUE_1, PlugInDataValue[TempSet][AUTO_VALUE_1], 1);

    if(PlugInDataValue[TempSet][AUTO_VALUE_2] != PlugInAutoDataValue[TempSet][AUTO_VALUE_2])
        DisplayPlugInValue(PlugInDC[TempSet], AUTO_VALUE_2, PlugInDataValue[TempSet][AUTO_VALUE_2], 1);

    if(PlugInDataValue[TempSet][AUTO_VALUE_3] != PlugInAutoDataValue[TempSet][AUTO_VALUE_3])
        DisplayPlugInValue(PlugInDC[TempSet], AUTO_VALUE_3, PlugInDataValue[TempSet][AUTO_VALUE_3], 1);

    if(PlugInDataValue[TempSet][AUTO_VALUE_4] != PlugInAutoDataValue[TempSet][AUTO_VALUE_4])
        DisplayPlugInValue(PlugInDC[TempSet], AUTO_VALUE_4, PlugInDataValue[TempSet][AUTO_VALUE_4], 1);

    if(PlugInDataValue[TempSet][AUTO_VALUE_5] != PlugInAutoDataValue[TempSet][AUTO_VALUE_5])
        DisplayPlugInValue(PlugInDC[TempSet], AUTO_VALUE_5, PlugInDataValue[TempSet][AUTO_VALUE_5], 1);

    if(PlugInDataValue[TempSet][AUTO_VALUE_6] != PlugInAutoDataValue[TempSet][AUTO_VALUE_6])
        DisplayPlugInValue(PlugInDC[TempSet], AUTO_VALUE_6, PlugInDataValue[TempSet][AUTO_VALUE_6], 1);
    }

return;
}
```

### > FX_RESYNC_CHASE_AUTOMATION

This function is called for all currently patched plug-ins whenever any track has been altered during *real-time* playback and needs automation values re-synchronized into the playback loop. Any track that contains automation data will send this message after the automation database is scanned to force all plug-ins to display and initialize any updated or changed controls at the end of the automation chase sequence to the current position.

The module should compare all current control values to the duplicate storage locations saved earlier. Then, the module should immediately update the display and re-initialize the control for each control value which has changed during the chase scan sequence.

If your module does not handle automation, simply ignore this call.

### In Parameters Used:

| | |
|---|---|
| *FX_Handler_DWord_Param1:* | Parameter Set |
| *FX_Handler_DWord_Param2:* | Track (zero based) |

### Out Parameters Used:

None.

### Other Active Parameters Set By SAWStudio:

None.

### Sample Code:

```
//=============================
// Global Variables Available
// To All Module Routines
//=============================

#define MAX_PARAM_SETS                  16
#define MAX_AUTO_CONTROLS               8

IQS_SAWFXDATA*              FxPtrSawData;

DWORD                       PlugInTrack[MAX_PARAM_SETS];
DWORD                       PlugInParamSetTable[MAX_PARAM_SETS];
HWND                        hWndPlugIn[MAX_PARAM_SETS];
HDC                         PlugInDC[MAX_PARAM_SETS];

int                         PlugInDataValue[MAX_PARAM_SETS][MAX_AUTO_CONTROLS];
int                         PlugInAutoDataValue[MAX_PARAM_SETS][MAX_AUTO_CONTROLS];

//============================
// SAWStudio Handler Routine
//
// ReSync Chase Automation
//============================

void    __stdcall   PlugInReSyncChaseAutomation(void)

{
// Temp Variables
```

```
//---------------
DWORD   TempParamSet;
DWORD   TempPhysicalTrack;

// Get Parameter Data
//-------------------
TempParamSet = FxPtrSawData->FX_Handler_DWord_Param1;

// Skip If Inactive
//-----------------
if(PlugInParamSetTable[TempParamSet] == 0)
    return;

TempPhysicalTrack = FxPtrSawData->FX_Handler_DWord_Param2;

// Verify A Track Match For This Parameter Set
//--------------------------------------------
if(TempPhysicalTrack != PlugInTrack[TempParamSet])
    return;

// Initialize And Display All Changed Values
//------------------------------------------
if(PlugInDataValue[TempParamSet][AUTO_BYPASS_SW] != PlugInAutoDataValue[TempParamSet][AUTO_BYPASS_SW])
    {
    InitPlugInVariables(TempParamSet, AUTO_BYPASS_SW);

    // Display If Window Is Open
    //--------------------------
    if(hWndPlugIn[TempParamSet])
        {
        if(PlugInDataValue[TempParamSet][AUTO_BYPASS_SW])
            DisplayBitmapBtn(PlugInDC[TempParamSet], &RectPlugInBypassDisplayBtn, hPlugInBypassBtnBitmap);

        else
            RefreshBitmap(PlugInDC[TempParamSet], &RectPlugInBypassDisplayBtn, hPlugInMainBitmap);
        }
    }

if(PlugInDataValue[TempParamSet][AUTO_VOL_POT] != PlugInAutoDataValue[TempParamSet][AUTO_VOL_POT])
    {
    InitPlugInVariables(TempParamSet, AUTO_VOL_POT);

    // Display If Window Is Open
    //--------------------------
    if(hWndPlugIn[TempParamSet])
        DrawVertKnob(PlugInDC[TempParamSet], &RectPlugInVolPot, PlugInDataValue[TempParamSet][AUTO_VOL_POT], 1);
    }

if(PlugInDataValue[TempParamSet][AUTO_VALUE_1] != PlugInAutoDataValue[TempParamSet][AUTO_VALUE_1])
    {
    InitPlugInVariables(TempParamSet, AUTO_VALUE_1);

    // Display If Window Is Open
    //--------------------------
```

```
        if(hWndPlugIn[TempParamSet])
            DisplayPlugInValue(PlugInDC[TempParamSet], AUTO_VALUE_1, PlugInDataValue[TempParamSet][AUTO_VALUE_1], 1);
        }

    if(PlugInDataValue[TempParamSet][AUTO_VALUE_2] != PlugInAutoDataValue[TempParamSet][AUTO_VALUE_2])
        {
        InitPlugInVariables(TempParamSet, AUTO_VALUE_2);

        // Display If Window Is Open
        //---------------------------
        if(hWndPlugIn[TempParamSet])
            DisplayPlugInValue(PlugInDC[TempParamSet], AUTO_VALUE_2, PlugInDataValue[TempParamSet][AUTO_VALUE_2], 1);
        }

    if(PlugInDataValue[TempParamSet][AUTO_VALUE_3] != PlugInAutoDataValue[TempParamSet][AUTO_VALUE_3])
        {
        InitPlugInVariables(TempParamSet, AUTO_VALUE_3);

        // Display If Window Is Open
        //---------------------------
        if(hWndPlugIn[TempParamSet])
            DisplayPlugInValue(PlugInDC[TempParamSet], AUTO_VALUE_3, PlugInDataValue[TempParamSet][AUTO_VALUE_3], 1);
        }

    if(PlugInDataValue[TempParamSet][AUTO_VALUE_4] != PlugInAutoDataValue[TempParamSet][AUTO_VALUE_4])
        {
        InitPlugInVariables(TempParamSet, AUTO_VALUE_4);

        // Display If Window Is Open
        //---------------------------
        if(hWndPlugIn[TempParamSet])
            DisplayPlugInValue(PlugInDC[TempParamSet], AUTO_VALUE_4, PlugInDataValue[TempParamSet][AUTO_VALUE_4], 1);
        }

    if(PlugInDataValue[TempParamSet][AUTO_VALUE_5] != PlugInAutoDataValue[TempParamSet][AUTO_VALUE_5])
        {
        InitPlugInVariables(TempParamSet, AUTO_VALUE_5);

        // Display If Window Is Open
        //---------------------------
        if(hWndPlugIn[TempParamSet])
            DisplayPlugInValue(PlugInDC[TempParamSet], AUTO_VALUE_5, PlugInDataValue[TempParamSet][AUTO_VALUE_5], 1);
        }

    if(PlugInDataValue[TempParamSet][AUTO_VALUE_6] != PlugInAutoDataValue[TempParamSet][AUTO_VALUE_6])
        {
        InitPlugInVariables(TempParamSet, AUTO_VALUE_6);

        // Display If Window Is Open
        //---------------------------
        if(hWndPlugIn[TempParamSet])
            DisplayPlugInValue(PlugInDC[TempParamSet], AUTO_VALUE_6, PlugInDataValue[TempParamSet][AUTO_VALUE_6], 1);
        }
```

```
return;
}
```

## > FX_CHANGE_PROCESS_AUTOMATION

This function is called for all currently patched plug-ins whenever an automation entry is encountered during any active process. Any track that encounters plug-in automation data during a process operation, sends this message to the designated plug-in to allow the plug-in to update and initialize its current data value for further processing at the new setting.

The module should immediately update and initialize the control value to its new value. The updates should not be displayed at this time, because this timing occurs in front of the actual playback position, but any processing after this message is received should take place with the new setting.

If your module does not handle automation, simply ignore this call.

### In Parameters Used:

*FX_Handler_DWord_Param1:* Parameter Set
*FX_Handler_DWord_Param2:* Track (zero based)
*FX_Handler_DWord_Param3:* Control ID#
*FX_Handler_int_Param1:* New Control Data Value

### Out Parameters Used:

None.

### Other Active Parameters Set By SAWStudio:

None.

### Sample Code:

```
//============================
// Global Variables Available
// To All Module Routines
//============================

#define MAX_PARAM_SETS                  16
#define MAX_AUTO_CONTROLS               8

IQS_SAWFXDATA*              FxPtrSawData;

DWORD                       PlugInTrack[MAX_PARAM_SETS];
DWORD                       PlugInParamSetTable[MAX_PARAM_SETS];

int                         PlugInDataValue[MAX_PARAM_SETS][MAX_AUTO_CONTROLS];

//===========================
// SAWStudio Handler Routine
//
// Change Process Automation
//=============================

void    __stdcall   PlugInChangeProcessAutomation(void)

{
DWORD   TempParamSet;
DWORD   TempPhysicalTrack;
DWORD   TempCtrlId;
int     TempDataValue;

// Get Parameter Data
//-------------------
TempParamSet = FxPtrSawData->FX_Handler_DWord_Param1;

// Skip If Inactive
//-----------------
if(PlugInParamSetTable[TempParamSet] == 0)
    return;

TempPhysicalTrack = FxPtrSawData->FX_Handler_DWord_Param2;

// Verify A Track Match For This Parameter Set
//-------------------------------------------
if(TempPhysicalTrack != PlugInTrack[TempParamSet])
    return;

// Set New Data Value And Initialize It
//------------------------------------
TempCtrlId = FxPtrSawData->FX_Handler_DWord_Param3;
TempDataValue = FxPtrSawData->FX_Handler_Int_Param1;

if(TempCtrlId == AUTO_BYPASS_SW)
    {
    PlugInDataValue[TempParamSet][AUTO_BYPASS_SW] = TempDataValue;

    InitPlugInVariables(TempParamSet, AUTO_BYPASS_SW);
    return;
    }

if(TempCtrlId == AUTO_VOL_POT)
    {
    PlugInDataValue[TempParamSet][AUTO_VOL_POT] = TempDataValue;

    InitPlugInVariables(TempParamSet, AUTO_VOL_POT);
    return;
    }

if(TempCtrlId == AUTO_VALUE_1)
    {
    PlugInDataValue[TempParamSet][AUTO_VALUE_1] = TempDataValue;
    InitPlugInVariables(TempParamSet, AUTO_VALUE_1);
    return;
    }

if(TempCtrlId == AUTO_VALUE_2)
    {
    PlugInDataValue[TempParamSet][AUTO_VALUE_2] = TempDataValue;

    InitPlugInVariables(TempParamSet, AUTO_VALUE_2);
    return;
    }

if(TempCtrlId == AUTO_VALUE_3)
    {
    PlugInDataValue[TempParamSet][AUTO_VALUE_3] = TempDataValue;

    InitPlugInVariables(TempParamSet, AUTO_VALUE_3);
    return;
    }

if(TempCtrlId == AUTO_VALUE_4)
    {
    PlugInDataValue[TempParamSet][AUTO_VALUE_4] = TempDataValue;

    InitPlugInVariables(TempParamSet, AUTO_VALUE_4);
    return;
    }

if(TempCtrlId == AUTO_VALUE_5)
    {
    PlugInDataValue[TempParamSet][AUTO_VALUE_5] = TempDataValue;

    InitPlugInVariables(TempParamSet, AUTO_VALUE_5);
    return;
    }

if(TempCtrlId == AUTO_VALUE_6)
    {
    PlugInDataValue[TempParamSet][AUTO_VALUE_6] = TempDataValue;

    InitPlugInVariables(TempParamSet, AUTO_VALUE_6);
    return;
    }

return;
}
```

### > FX_DISPLAY_PROCESS_AUTOMATION

This function is called for all currently patched plug-ins whenever an automation entry is encountered during a *real-time* playback process. Any track that encounters plug-in automation data during a real-time playback process , sends this message in sync with the playback position to allow the plug-in to update its display of changed control values.

The module should immediately update the display for the designated control.

If your module does not handle automation, simply ignore this call.

### In Parameters Used:

| | |
|---|---|
| *FX_Handler_DWord_Param1:* | Parameter Set |
| *FX_Handler_DWord_Param2:* | Track (zero based) |
| *FX_Handler_DWord_Param3:* | Control ID# |
| *FX_Handler_int_Param1:* | New Control Data Value |

### Out Parameters Used:

None.

### Other Active Parameters Set By SAWStudio:

None.

### Sample Code:

```
//============================
// Global Variables Available
// To All Module Routines
//============================

#define MAX_PARAM_SETS                  16
#define MAX_AUTO_CONTROLS               8

IQS_SAWFXDATA*              FxPtrSawData;

DWORD                       PlugInTrack[MAX_PARAM_SETS];
DWORD                       PlugInParamSetTable[MAX_PARAM_SETS];
HWND                        hWndPlugIn[MAX_PARAM_SETS];
HDC                         PlugInDC[MAX_PARAM_SETS];

//============================
// SAWStudio Handler Routine
//
// Display Process Automation
//============================

void    __stdcall   PlugInDisplayProcessAutomation(void)

{
DWORD   TempParamSet;
DWORD   TempPhysicalTrack;
DWORD   TempCtrlId;
int     TempDataValue;
```

```
// Get Parameter Data
//-------------------
TempParamSet = FxPtrSawData->FX_Handler_DWord_Param1;

// Skip If Inactive
//-----------------
if(PlugInParamSetTable[TempParamSet] == 0)
    return;

// Skip If Window Is Not Open
//---------------------------
if(hWndPlugIn[TempParamSet] == 0)
    return;

TempPhysicalTrack = FxPtrSawData->FX_Handler_DWord_Param2;

// Verify A Track Match For This Parameter Set
//--------------------------------------------
if(TempPhysicalTrack != PlugInTrack[TempParamSet])
    return;

// Display Automation Changes
//---------------------------
TempCtrlId = FxPtrSawData->FX_Handler_DWord_Param3;
TempDataValue = FxPtrSawData->FX_Handler_Int_Param1;

if(TempCtrlId == AUTO_BYPASS_SW)
    {
    if(TempDataValue)
        DisplayBitmapBtn(PlugInDC[TempParamSet], &RectPlugInBypassDisplayBtn, hPlugInBypassBtnBitmap);
    else
        RefreshBitmap(PlugInDC[TempParamSet], &RectPlugInBypassDisplayBtn, hPlugInMainBitmap);

    return;
    }

if(TempCtrlId == AUTO_VOL_POT)
    {
    DrawVertKnob(PlugInDC[TempParamSet], &RectPlugInVolPot, TempDataValue, 1);
    return;
    }

if(TempCtrlId == AUTO_VALUE_1)
    {
    DisplayPlugInValue(PlugInDC[TempParamSet], AUTO_VALUE_1, TempDataValue, 1);
    return;
    }

if(TempCtrlId == AUTO_VALUE_2)
    {
    DisplayPlugInValue(PlugInDC[TempParamSet], AUTO_VALUE_2, TempDataValue, 1);
    return;
    }
```

```
if(TempCtrlId == AUTO_VALUE_3)
    {
    DisplayPlugInValue(PlugInDC[TempParamSet], AUTO_VALUE_3, TempDataValue, 1);
    return;
    }

if(TempCtrlId == AUTO_VALUE_4)
    {
    DisplayPlugInValue(PlugInDC[TempParamSet], AUTO_VALUE_4, TempDataValue, 1);
    return;
    }

if(TempCtrlId == AUTO_VALUE_5)
    {
    DisplayPlugInValue(PlugInDC[TempParamSet], AUTO_VALUE_5, TempDataValue, 1);
    return;
    }

if(TempCtrlId == AUTO_VALUE_6)
    {
    DisplayPlugInValue(PlugInDC[TempParamSet], AUTO_VALUE_6, TempDataValue, 1);
    return;
    }

return;
}
```

## 3) FX Process Buffer Function

**void __stdcall FxProcessBuffer (DWORD FxParamSet, LPBYTE FxBuffPtr)**

| | |
|---|---|
| *DWORD FxParamSet:* | Current Parameter Set Reference For This Call |
| *LPBYTE FxBuffPtr:* | Pointer To Current Data Buffer |

This function is called with each buffer of data needing to be processed. This is the function that actually does the work of altering the audio data. Note that speed of execution in this function is extremely important, and the fastest code you can write (preferably assembly language) should be used here.

The parameters passed include the current *Parameter Set* to be worked with for this call, and the *Buffer Pointer* where the module will find the audio data needing processing.

The *FX_BufferByteSize* Shared Memory Variable will contain the buffer size in bytes

The data format is derived from the *FX_BufferChans* and *FX_BufferBytesPerSample* Shared Memory Variables. The bytes per sample times the number of channels will equal the total number of bytes between samples. In most patches, **SAWStudio** will pass 32 bit stereo data (DWORD Per Sample, Sign Extended 24 Bit Aligned Data) which will contain 4 bytes per sample and 2 channels. In *Output Track Final Res* patches, 16 Bit Stereo formats will be common, containing 2 bytes per sample and 2 channels. The channel data is interleaved *Left* then *Right*. The module should be capable of processing variable formats. If a format is passed that is not supported by the module, it should simply return, leaving the buffer data intact.

The data buffer should be processed in-place, returning the processed data in the same buffer. If you use extra work buffers during the process, make sure that the final results are placed back into the original buffer.

The DWORD format for 24 bit data can be processed as straight DWORD values. Actually the values are signed so typical C code would label them as INTEGERS. Because of the 24 bit alignment, you should not have to worry about clipping overhead as there are 8 bits of headroom, which should handle most any situation. If you prefer to work in floating point, a simple memory copy from the data buffer to a work buffer declared as FLOAT is all that is needed. Process the data and copy the results back to the original buffer as integers.

You may alter the size of the buffer freely as long as you do not exceed the *FX_MaxBufferByteSize* Shared Memory Variable size. If you alter the size, you must reset the *FX_BufferByteSize* Shared Memory Variable to the new size, letting **SAWStudio** know how much data in the buffer is valid.

You may also copy and store the data in your own internal buffers and return zero bytes, effectively withholding the data until you collect multiple buffers for complex algorithm processing.

If you perform any metering displays, the clipping reference for the 16 bit data are the values 0x7FFF and 0x8001. Clipping references for 24 bit data are 0x007FFFFF and 0xFF800001.

If your module requires accurate position information regarding this current buffer's position in the MultiTrack, you should reference the *FX_CurProcessPos* Shared Memory Variable. This is the actual relative position of the current buffer's first sample of data within the MultiTrack environment. Since buffer data is processed in front of the actual playback position, the actual buffer position you are currently processing will have no relation to the actual audio currently being heard. If your plug-in displays meter information, the meter displays will not accurately match the audio playback. This has been a problem with most other plug-in API protocols. The **SAWStudio** API handles this situation by also passing accurate playback position information through a separate hi resolution timing function call. By storing your meter calculation results at this time along with the *FX_CurProcessPos* and displaying them later during the playback position function call, you can display absolutely accurate reference meters in perfect sync with the playback audio.

## Sample Code:

```
//============================
// Global Variables Available
// To All Module Routines
//============================

#define MAX_PARAM_SETS                  16
#define MAX_AUTO_CONTROLS               8

IQS_SAWFXDATA*              FxPtrSawData;

DWORD                       FxProcessParamSet = 0;
DWORD                       FxProcessBytesPerSample = 4;
LPBYTE                      FxProcessBuffPtr;
DWORD                       FxProcessBuffSize;
DWORD                       FxProcessPos;

int                         PlugInDataValue[MAX_PARAM_SETS][MAX_AUTO_CONTROLS];
```

```
//======================
// SAWStudio API Routine
//
// FX Process Buffer
//======================

void  __stdcall  FxProcessBuffer(DWORD ParamSet, LPBYTE BuffPtr)

{
// Set Process Variables For This Call
//------------------------------------
FxProcessParamSet = ParamSet;
FxProcessBuffPtr = BuffPtr;
FxProcessBuffSize = FxPtrSawData->FX_BufferByteSize;
FxProcessBytesPerSample = FxPtrSawData->FX_BufferBytesPerSample;
FxProcessPos = FxPtrSawData->FX_CurProcessPos;

// This Plug-In Only Processes Stereo Buffers
//-------------------------------------------
if(FxPtrSawData->FX_BufferChans != 2)
    return;

// Skip Process If PlugIn Is Bypassed
//-----------------------------------
if(PlugInDataValue[FxProcessParamSet][AUTO_BYPASS_SW])
    return;

// Process The Correct Buffer Format
//----------------------------------
if(FxProcessBytesPerSample == 4)
    DoPlugInProcess32();
else
    DoPlugInProcess16();

// Set Final BufferSize If Your Process Changes It
// Note That The Size Can Not Grow Beyond FxMaxBufferSize
//-------------------------------------------------------
FxPtrSawData->FX_BufferByteSize = FxProcessBuffSize;

return;
}
```

## 4) FX Change Position Function

**void __stdcall FxChangePosition (void)**

This function is called whenever the MultiTrack cursor sample position changes. During *real-time* playback, this maintains an accurate lock to the audio device output.

This call is made once per timing reference to all patched plug-ins. It is up to the plug-in itself to adjust all active parameter sets if needed.

If your plug-in does not need this position information, simply return immediately.

This routine is called from a **Time Critical Priority** thread, so you must process the data as fast as possible and return to avoid dragging the system down.

The *FX_CurSamplePos* Shared Memory Variable contains the new sample position. This is the actual playback sample position that is currently being heard by the user.

If your module displays *real-time* data, this call can be used to update your display from the stored data that was calculated during the *FxProcessBuffer* function. Compare this position with the stored process position to display the proper data at the proper time.

The *FX_CurSmpteFrameCount* Shared Memory Variable contains the total Smpte frame count value.

The *FX_CurSmpteSFrames* Shared Memory Variable contains the Smpte sub frame value.

The *FX_CurSmpteFrames* Shared Memory Variable contains the Smpte frame value.

The *FX_CurSmpteSecs* Shared Memory Variable contains the Smpte seconds value.

The *FX_CurSmpteMins* Shared Memory Variable contains the Smpte minutes value.

The *FX_CurSmpteHours* Shared Memory Variable contains the Smpte Hours value.

## Sample Code:

```
//==============================
// Global Variables Available
// To All Module Routines
//==============================

IQS_SAWFXDATA*              FxPtrSawData;

DWORD                       FxSamplePos;

//=========================
// SAWStudio API Routine
//
// FX Change Position
//=========================

void  __stdcall  FxChangePosition(void)

{
// Get Parameter Data
//--------------------
FxSamplePos = FxPtrSawData->FX_CurSamplePos;

// Display All Meters At Current Sample Position Reference
//-------------------------------------------------------
DisplayAllMeters();
return;
}
```

## FX EXECUTE FUNCTION MESSAGE

The API uses a single communication message which the module may register to request **SAWStudio** to execute certain functions. The message is registered as follows:

```
char            szSawFxExecuteFunctionMsg[] = "SawFxExecuteFunction";
DWORD           SawFxExecuteFunctionMsg = 0;

SawFxExecuteFunctionMsg = RegisterWindowMessage(szSawFxExecuteFunctionMsg);
```

Any time the plug-in wishes to execute one of the supported functions, it sends the message, including the proper function ID value, to **SAWStudio** after pre-setting the proper *FX_Function* parameters.

```
SendMessage(hWndSawMain, SawFxExecuteFunctionMsg, FUNCTION_ID, LParam);
```

The supported function ID's are defined in the Header File as follows:

```
#define FORCE_FX_PRESCAN_STOP               1
#define FORCE_FX_PRESCAN_CONTINUE           2
#define MARK_MT_BEG                         3
#define MARK_MT_END                         4
#define CLEAR_MT_MARKS                      5
#define SPLIT_KEEP_MTENTRY                  6
#define SPLIT_MARK_MTENTRY                  7
#define SPLIT_REMOVE_MTENTRY                8
#define SPLIT_DELETE_MTENTRY                9
#define SELECT_MODE_ON                      10
#define SELECT_MODE_OFF                     11
#define EDL_UNDO_OFF                        12
#define EDL_UNDO_ON                         13
#define EDL_SAVE_UNDO                       14
#define EDL_RECALL_UNDO                     15
```

```
#define REFRESH_FX_DATA                   16

#define SET_MT_SAMPLE_POS                 17
#define SET_MT_SMPTE_FRAME_POS        18

#define START_MT_AUDIO_PLAY               19
#define START_MT_AUDIO_REC                20
#define START_MT_AUDIO_SRP                21
#define STOP_MT_AUDIO                     22

#define SET_EDL_MODIFY_FLAG               23
#define CHANGE_MT_ACTIVE_TRACK            24

#define SET_MAX_READ_AHEAD                25
#define SET_MIN_READ_AHEAD                26
#define REFRESH_HOST_DISPLAY              27

#define SET_SOLO_SWITCH_VALUE             50

#define SEND_TO_AUTOMATION                100
```

### > FORCE_FX_PRESCAN_STOP

This function will force **SAWStudio** to pre-scan the MultiTrack and then stop. During the pre-scan, data buffers are sent to the patched plug-ins for the purpose of gathering data information.

**Parameters Set By Plug-in:**
None.

**LParam Passed In Message:**
0

**Parameters Returned By SAWStudio:**
None.

**Sample Code:**
```
SendMessage(hWndSawMain, SawFxExecuteFunctionMsg, FORCE_FX_PRESCAN_STOP, 0);
```

### > FORCE_FX_PRESCAN_CONTINUE

This function will force **SAWStudio** to pre-scan the MultiTrack and then continue to playback. During the pre-scan, data buffers are sent to the patched plug-ins for the purpose of gathering data information.

**Parameters Set By Plug-in:**

None.

**LParam Passed In Message:**

0

**Parameters Returned By SAWStudio:**

None.

**Sample Code:**

```
SendMessage(hWndSawMain, SawFxExecuteFunctionMsg, FORCE_FX_PRESCAN_CONTINUE, 0);
```

### > MARK_MT_BEG

This function will cause **SAWStudio** to set the current MT cursor position as the mark begin point.

**Parameters Set By Plug-in:**

None.

**LParam Passed In Message:**

0

**Parameters Returned By SAWStudio:**

None.

**Sample Code:**

```
SendMessage(hWndSawMain, SawFxExecuteFunctionMsg, MARK_MT_BEG, 0);
```

### > MARK_MT_END

This function will cause **SAWStudio** to set the current MT cursor position as the mark end point.

**Parameters Set By Plug-in:**

None.

### LParam Passed In Message:
0

### Parameters Returned By SAWStudio:
None.

### Sample Code:
```
SendMessage(hWndSawMain, SawFxExecuteFunctionMsg, MARK_MT_END, 0);
```

## > CLEAR_MT_MARKS
This function will cause **SAWStudio** to clear the current MT mark begin and end points.

### Parameters Set By Plug-in:
None.

### LParam Passed In Message:
0

### Parameters Returned By SAWStudio:
None.

### Sample Code:
```
SendMessage(hWndSawMain, SawFxExecuteFunctionMsg, CLEAR_MT_MARKS, 0);
```

## > SPLIT_KEEP_MTENTRY
This function will cause **SAWStudio** to split the entry at the current MT cursor position, and keep both new entries.

### Parameters Set By Plug-in:
None.

### LParam Passed In Message:
0

### Parameters Returned By SAWStudio:
None.

### Sample Code:

```
SendMessage(hWndSawMain, SawFxExecuteFunctionMsg, SPLIT_KEEP_MTENTRY, 0);
```

### > SPLIT_MARK_MTENTRY

This function will cause **SAWStudio** to split the entry at the current MT cursor position, and mark the left new entry for use in Select Mode.

### Parameters Set By Plug-in:

None.

### LParam Passed In Message:

0

### Parameters Returned By SAWStudio:

None.

### Sample Code:

```
SendMessage(hWndSawMain, SawFxExecuteFunctionMsg, SPLIT_MARK_MTENTRY, 0);
```

### > SPLIT_REMOVE_MTENTRY

This function will cause **SAWStudio** to split the entry at the current MT cursor position, and remove the left new entry without disturbing the following entries.

### Parameters Set By Plug-in:

None.

### LParam Passed In Message:

0

### Parameters Returned By SAWStudio:

None.

### Sample Code:

```
SendMessage(hWndSawMain, SawFxExecuteFunctionMsg, SPLIT_REMOVE_MTENTRY, 0);
```

### > SPLIT_DELETE_MTENTRY

This function will cause **SAWStudio** to split the entry at the current MT cursor position, and delete the left new entry pulling the following butt-spliced entries forward.

### Parameters Set By Plug-in:

None.

### LParam Passed In Message:
0

### Parameters Returned By SAWStudio:
None.

### Sample Code:
```
SendMessage(hWndSawMain, SawFxExecuteFunctionMsg, SPLIT_DELETE_MTENTRY, 0);
```

## > SELECT_MODE_ON
This function will cause **SAWStudio** to activate Select Mode.

### Parameters Set By Plug-in:
None.

### LParam Passed In Message:
0

### Parameters Returned By SAWStudio:
None.

### Sample Code:
```
SendMessage(hWndSawMain, SawFxExecuteFunctionMsg, SELECT_MODE_ON, 0);
```

## > SELECT_MODE_OFF
This function will cause **SAWStudio** to de-activate Select Mode.

### Parameters Set By Plug-in:
None.

### LParam Passed In Message:
0

### Parameters Returned By SAWStudio:
None.

### Sample Code:

```
SendMessage(hWndSawMain, SawFxExecuteFunctionMsg, SELECT_MODE_OFF, 0);
```

## > EDL_UNDO_OFF

This function will cause **SAWStudio** to bypass automatic undo file creation until the EDL_UNDO_ON function is activated. This option can be used to stop the creation of multiple undo files during a series of operations that would normally each create undo files on their own.

### Parameters Set By Plug-in:

None.

### LParam Passed In Message:

0

### Parameters Returned By SAWStudio:

None.

### Sample Code:

```
SendMessage(hWndSawMain, SawFxExecuteFunctionMsg, EDL_UNDO_OFF, 0);
```

## > EDL_UNDO_ON

This function will cause **SAWStudio** to allow automatic undo file creation again.

### Parameters Set By Plug-in:

None.

### LParam Passed In Message:

0

### Parameters Returned By SAWStudio:

None.

### Sample Code:

```
SendMessage(hWndSawMain, SawFxExecuteFunctionMsg, EDL_UNDO_ON, 0);
```

## > EDL_SAVE_UNDO

This function will cause **SAWStudio** to create an undo file immediately, saving the current state of the EDL.

### Parameters Set By Plug-in:
None.

### LParam Passed In Message:
0

### Parameters Returned By SAWStudio:
None.

### Sample Code:
```
SendMessage(hWndSawMain, SawFxExecuteFunctionMsg, EDL_SAVE_UNDO, 0);
```

## > EDL_RECALL_UNDO
This function will cause **SAWStudio** to recall the last undo file immediately, restoring the EDL to that saved condition.

### Parameters Set By Plug-in:
None.

### LParam Passed In Message:
0

### Parameters Returned By SAWStudio:
None.

### Sample Code:
```
SendMessage(hWndSawMain, SawFxExecuteFunctionMsg, EDL_RECALL_UNDO, 0);
```

## > REFRESH_FX_DATA
This function will cause **SAWStudio** to refresh various Shared Memory Variables to current values based on the defined ID value passed in the LParam parameter.

The supported ID's are defined in the Header File as follows:

**#define FX_WINDOW_DATA 0x00000001**
**#define FX_SMPTE_DATA 0x00000002**
**#define FX_TEMPO_DATA 0x00000004**

** ***FX_WINDOW_DATA***

**Parameters Set By Plug-in:**
None.

**LParam Passed In Message:**
LPARAM FX_WINDOW_DATA

**Parameters Returned By SAWStudio:**
*PtrSawFxData->FX_hWndMultiTrack*
*PtrSawFxData->FX_MTSampleRate*
*PtrSawFxData->FX_MTResolution*
*PtrSawFxData->FX_MTZoomX*
*PtrSawFxData->FX_MTZoomY*
*PtrSawFxData->FX_MTMarkBegPos*
*PtrSawFxData->FX_MTMarkEndPos*

*PtrSawFxData->FX_MTWaveDBScale*
*PtrSawFxData->FX_MTWaveCenterLft*
*PtrSawFxData->FX_MTWaveCenterRgt*
*PtrSawFxData->FX_MTWaveDivider*
*PtrSawFxData->FX_MTWaveTop*
*PtrSawFxData->FX_MTWaveBot*
*PtrSawFxData->FX_MTWaveLft*
*PtrSawFxData->FX_MTWaveRgt*
*PtrSawFxData->FX_MTZoomShiftY*
*PtrSawFxData->FX_MTWaveTrack*
*PtrSawFxData->FX_MTStartSamplePos*
*PtrSawFxData->FX_MTCursorSamplePos*
*PtrSawFxData->FX_MTWaveOkToDraw*

*PtrSawFxData->FX_hWndSoundFile*
*PtrSawFxData->FX_SFSampleRate*
*PtrSawFxData->FX_SFResolution*
*PtrSawFxData->FX_SFZoomX*
*PtrSawFxData->FX_SFZoomY*

*PtrSawFxData->FX_SFZoomShiftY*
*PtrSawFxData->FX_SFZoomXMagnify*
*PtrSawFxData->FX_SFMarkBegPos*
*PtrSawFxData->FX_SFMarkEndPos*
*PtrSawFxData->FX_SFStartSamplePos*
*PtrSawFxData->FX_SFCursorSamplePos*

*PtrSawFxData->FX_SFWaveDBScale*
*PtrSawFxData->FX_SFWaveCenterLft*
*PtrSawFxData->FX_SFWaveCenterRgt*
*PtrSawFxData->FX_SFWaveDivider*
*PtrSawFxData->FX_SFWaveTop*
*PtrSawFxData->FX_SFWaveBot*
*PtrSawFxData->FX_SFWaveLft*
*PtrSawFxData->FX_SFWaveRgt*
*PtrSawFxData->FX_SFWaveOkToDraw*

**Sample Code:**

```
SendMessage(hWndSawMain, SawFxExecuteFunctionMsg, SET_MT_SAMPLE_POS, FX_WINDOW_DATA);
```

### ** *FX_SMPTE_DATA*

**Parameters Set By Plug-in:**

None.

**LParam Passed In Message:**

LPARAM FX_SMPTE_DATA

**Parameters Returned By SAWStudio:**

*PtrSawFxData->FX_CurSmpteStartOffsetHours*
*PtrSawFxData->FX_CurSmpteStartOffsetMins*
*PtrSawFxData->FX_CurSmpteStartOffsetSecs*
*PtrSawFxData->FX_CurSmpteStartOffsetFrames*
*PtrSawFxData->FX_CurSmpteStartOffsetSFrames*
*PtrSawFxData->FX_CurSmpteFormat*
*PtrSawFxData->FX_CurSmpteMode*

**Sample Code:**

```
SendMessage(hWndSawMain, SawFxExecuteFunctionMsg, SET_MT_SAMPLE_POS, FX_SMPTE_DATA);
```

### ** *FX_TEMPO_DATA*

**Parameters Set By Plug-in:**
None.

**LParam Passed In Message:**
LPARAM FX_TEMPO_DATA

**Parameters Returned By SAWStudio:**
*PtrSawFxData->FX_CurTempoBeatsPerMin*
*PtrSawFxData->FX_CurTempoBeatsPerMeasure*
*PtrSawFxData->FX_CurTempoBeatRes*
*PtrSawFxData->FX_CurTempoTickRes*

**Sample Code:**

```
SendMessage(hWndSawMain, SawFxExecuteFunctionMsg, SET_MT_SAMPLE_POS, FX_TEMPO_DATA);
```

### > SET_MT_SAMPLE_POS

This function will cause **SAWStudio** to set and locate the MultiTrack Cursor to a desired sample position.

**Parameters Set By Plug-in:**
None.

**LParam Passed In Message:**
LPARAM SamplePosition

**Parameters Returned By SAWStudio:**
None.

**Sample Code:**

```
SendMessage(hWndSawMain, SawFxExecuteFunctionMsg, SET_MT_SAMPLE_POS, SamplePosition);
```

### > SET_MT_SMPTE_FRAME_POS

This function will cause **SAWStudio** to set and locate the MultiTrack Cursor to a desired Smpte Frame Count position. This position reference is a total frame count reference, not a Smpte Time reference.

**Parameters Set By Plug-in:**

None.

**LParam Passed In Message:**

LPARAM FrameCount

**Parameters Returned By SAWStudio:**

None.

**Sample Code:**

```
SendMessage(hWndSawMain, SawFxExecuteFunctionMsg, SET_MT_SMPTE_FRAME_POS, FrameCount);
```

### > START_MT_AUDIO_PLAY

This function will cause **SAWStudio** to start MultiTrack playback at the current cursor position.

**Parameters Set By Plug-in:**

None.

**LParam Passed In Message:**

0

**Parameters Returned By SAWStudio:**

None.

**Sample Code:**

```
SendMessage(hWndSawMain, SawFxExecuteFunctionMsg, START_MT_AUDIO_PLAY, 0);
```

### > START_MT_AUDIO_REC

This function will cause **SAWStudio** to start MultiTrack record at the current cursor position.

**Parameters Set By Plug-in:**

None.

**LParam Passed In Message:**
0

**Parameters Returned By SAWStudio:**
None.

**Sample Code:**
```
SendMessage(hWndSawMain, SawFxExecuteFunctionMsg, START_MT_AUDIO_REC, 0);
```

### > START_MT_AUDIO_SRP

This function will cause **SAWStudio** to start MultiTrack SRP record/playback at the current cursor position.

**Parameters Set By Plug-in:**
None.

**LParam Passed In Message:**
0

**Parameters Returned By SAWStudio:**
None.

**Sample Code:**
```
SendMessage(hWndSawMain, SawFxExecuteFunctionMsg, START_MT_AUDIO_SRP, 0);
```

### > STOP_MT_AUDIO

This function will cause **SAWStudio** to stop MultiTrack playback or record operation.

**Parameters Set By Plug-in:**
None.

**LParam Passed In Message:**
0

**Parameters Returned By SAWStudio:**
None.

**Sample Code:**

```
SendMessage(hWndSawMain, SawFxExecuteFunctionMsg, STOP_MT_AUDIO, 0);
```

### > SET_EDL_MODIFY_FLAG

This function will cause **SAWStudio** to set the EDL Modified flag, which causes a warning to save the EDL whenever the EDL is closed, changed, or the program is about to be shutdown.

**Parameters Set By Plug-in:**
None.

**LParam Passed In Message:**
0

**Parameters Returned By SAWStudio:**
None.

**Sample Code:**
```
SendMessage(hWndSawMain, SawFxExecuteFunctionMsg, SET_EDL_MODIFY_FLAG, 0);
```

### > CHANGE_MT_ACTIVE_TRACK

This function will cause **SAWStudio** to change the MultiTrack Active Track to the desired track.

**Parameters Set By Plug-in:**
None.

**LParam Passed In Message:**

| | |
|---|---|
| LPARAM | PhysicalTrack (zero based) |

**Parameters Returned By SAWStudio:**
None.

**Sample Code:**
```
SendMessage(hWndSawMain, SawFxExecuteFunctionMsg, CHANGE_MT_ACTIVE_TRACK, PhysicalTrack);
```

### > SET_MAX_READ_AHEAD

This function will cause **SAWStudio** to set the maximum read-ahead flag which causes the dynamic latency engine to expand its latency limits temporarily.

While this flag is set, the engine builds multitrack data in advance of the normal latency limits, thus allowing intense background operations to be performed without glitching the audio playback. Use of this function should always be followed by the inverse function when the intensive background operation is complete, so the latency limits are set back to *real-time*.

### Parameters Set By Plug-in:
None.

### LParam Passed In Message:
0

### Parameters Returned By SAWStudio:
None.

### Sample Code:
```
SendMessage(hWndSawMain, SawFxExecuteFunctionMsg, SET_MAX_READ_AHEAD, 0);
```

## > SET_MIN_READ_AHEAD
This function will cause **SAWStudio** to set the minimum read-ahead flag which causes the dynamic latency engine to compress its latency limits back to its low *real-time* settings.

While this flag is set, the engine builds multitrack data directly in front of the audio playback maintaining its normal *real-time* latency limits.

### Parameters Set By Plug-in:
None.

### LParam Passed In Message:
0

### Parameters Returned By SAWStudio:
None.

### Sample Code:
```
SendMessage(hWndSawMain, SawFxExecuteFunctionMsg, SET_MIN_READ_AHEAD, 0);
```

## > REFRESH_HOST_DISPLAY
This function will cause **SAWStudio** to refresh all of its visible windows. This function can be used when a plug-in removes a large display window from the screen to force **SAWStudio** to update its display immediately, instead of waiting for Windows to do the update in its own time frame, possibly leaving large blank screen areas onscreen for large amounts of time.

### Parameters Set By Plug-in:
None.

### LParam Passed In Message:
0

### Parameters Returned By SAWStudio:
None.

### Sample Code:
```
SendMessage(hWndSawMain, SawFxExecuteFunctionMsg, REFRESH_HOST_DISPLAY, 0);
```

## > SET_SOLO_SWITCH_VALUE
This function will cause **SAWStudio** to set the Solo Switch on or off for a specific track/channel.

### Parameters Set By Plug-in:
| | |
|---|---|
| *FX_Track* | Track (zero based) |

### Parameters Returned By SAWStudio:
None.

### LParam Passed In Message:
| | |
|---|---|
| LPARAM | SwitchValue (0 = Off) (1 = On) |

### Sample Code:
```
SendMessage(hWndSawMain, SawFxExecuteFunctionMsg, SET_SOLO_SWITCH_VALUE, SwitchValue);
```

## > SEND_TO_AUTOMATION
This function will cause **SAWStudio** to store control data to its automation database. The *FX_MTAutomationWriteFlag* should be checked to find out if *Automation Write Mode* is active. If this value is zero, then the control data should be stored as a new default value instead of sent to **SAWStudio**.

### Parameters Set By Plug-in:
| | |
|---|---|
| *FX_Handler_DWord_Param1:* | Plug-In Index Value |
| *FX_Handler_DWord_Param2:* | Parameter Set |
| *FX_Handler_DWord_Param3:* | Track (zero based) |
| *FX_Handler_DWord_Param4:* | Control ID# |

*FX_Handler_int_Param1:* Control Data Value
*FX_Handler_int_Param2:* Control Min Data Value
*FX_Handler_int_Param3:* Control Max Data Value

### LParam Passed In Message:

0

### Parameters Returned By SAWStudio:

None.

### Sample Code:

```
//============================
// Global Variables Available
// To All Module Routines
//============================

#define MAX_PARAM_SETS                  16
#define MAX_AUTO_CONTROLS               8

IQS_SAWFXDATA*              FxPtrSawData;

DWORD                       FxProcessParamSet = 0;
DWORD                       FxProcessBytesPerSample = 4;
LPBYTE                      FxProcessBuffPtr;
DWORD                       FxProcessBuffSize;
DWORD                       FxProcessPos;

int                         PlugInDefaultDataValue[MAX_PARAM_SETS][MAX_AUTO_CONTROLS];

//=================
// Send Automation
//=================

void    __stdcall   SendAutomation(DWORD TempCtrlId, int TempDataValue, int TempMinDataValue, int TempMaxDataValue)

{
// Set As New Default Value If Automation Mode Is Not Active
//----------------------------------------------------------
if(FxPtrSawData->FX_MTAutomationWriteFlag == 0)
    {
    PlugInDefaultDataValue[FxActiveParamSet][TempCtrlId] = TempDataValue;
    return;
    }

// Send To Automation
//-------------------
FxPtrSawData->FX_Function_DWord_Param1 = FxPlugInIndex;
FxPtrSawData->FX_Function_DWord_Param2 = FxActiveParamSet;
FxPtrSawData->FX_Function_DWord_Param3 = PlugInTrack[FxActiveParamSet];
FxPtrSawData->FX_Function_DWord_Param4 = TempCtrlId;
FxPtrSawData->FX_Function_Int_Param1 = TempDataValue;
FxPtrSawData->FX_Function_Int_Param2 = TempMinDataValue;
FxPtrSawData->FX_Function_Int_Param3 = TempMaxDataValue;

SendMessage(hWndSawMain, SawFxExecuteFunctionMsg, SEND_TO_AUTOMATION, 0);
return;
}
```

## A DETAILED WALK-THROUGH OF THE TEST PLUG-IN SHELL

Here is a listing of the Test Plug-In Shell code. Some of the routines that are not directly tied to understanding the API have been left out. This section will walk through the code and explain, in detail, the workings of the module and its connection to the API. All of the code has been written in straight C, although the file is a .cpp type, allowing you to write C++ code as well. Much of the code has been expanded out for clarity and is not necessarily optimized for best performance. The concepts used here may be followed as a guideline and expanded upon for your own needs, or you might care to learn the basics of the API operation and write the code completely in your own style.

This test module will demonstrate the basics of modifying the processing buffers, displaying an interface to the user, hooking into the automation engine, and displaying accurately synchronized meter displays.

### FX_API_TestPlugIn.h file.

**We start with the SAWFXDATA structure declaration. This is the main variable and parameter link between the .dll and the main host program.**

```
//===================================
// SAWStudio FX API TestPlugIn - DLL
//
// STRUCTURE DECLARATIONS
// EQUATES
// PROCEDURE DECLARATIONS
// GLOBAL VARIABLES
//
// copyright 2001 By Bob Lentini
//===================================

//========================
// STRUCTURE DECLARATIONS
//========================

// FX Shared Memory Data Structure
//--------------------------------
typedef struct
    {
    HWND            FX_hWndMain;
```

```
HFONT           FX_IQSSysFont;
DWORD           FX_VersionNum;

DWORD           FX_Handler_DWord_Param1;
DWORD           FX_Handler_DWord_Param2;
DWORD           FX_Handler_DWord_Param3;
DWORD           FX_Handler_DWord_Param4;

int             FX_Handler_Int_Param1;
int             FX_Handler_Int_Param2;
int             FX_Handler_Int_Param3;
int             FX_Handler_Int_Param4;

LPBYTE          FX_Handler_Ptr_Param1;
LPBYTE          FX_Handler_Ptr_Param2;
LPBYTE          FX_Handler_Ptr_Param3;
LPBYTE          FX_Handler_Ptr_Param4;

DWORD           FX_Function_DWord_Param1;
DWORD           FX_Function_DWord_Param2;
DWORD           FX_Function_DWord_Param3;
DWORD           FX_Function_DWord_Param4;

int             FX_Function_Int_Param1;
int             FX_Function_Int_Param2;
int             FX_Function_Int_Param3;
int             FX_Function_Int_Param4;

LPBYTE          FX_Function_Ptr_Param1;
LPBYTE          FX_Function_Ptr_Param2;
LPBYTE          FX_Function_Ptr_Param3;
LPBYTE          FX_Function_Ptr_Param4;

DWORD           FX_MultiTrackRtnTrackOffset;
DWORD           FX_MultiTrackOutTrackOffset;
DWORD           FX_MultiTrackVideoTrackOffset;
DWORD           FX_MultiTrackControlTrackOffset;
DWORD           FX_MultiTrackLastTrackOffset;

HWND            FX_hWndMultiTrack;
DWORD           FX_MTSampleRate;
DWORD           FX_MTResolution;
DWORD           FX_MTZoomX;
DWORD           FX_MTZoomY;
DWORD           FX_MTMarkBegPos;
DWORD           FX_MTMarkEndPos;

char            FX_Title[48];
DWORD           FX_PlugInIndex;
DWORD           FX_ParamSet;
DWORD           FX_Track;
DWORD           FX_BufferRatioIn;
DWORD           FX_BufferRatioOut;
DWORD           FX_RequestXtra;
DWORD           FX_RequestCancel;

DWORD           FX_ProcessFlag;
DWORD           FX_BufferByteSize;
DWORD           FX_BufferChans;
DWORD           FX_BufferBytesPerSample;
DWORD           FX_MaxBufferByteSize;

DWORD           FX_CurProcessPos;
DWORD           FX_CurSamplePos;
DWORD           FX_ProcessBegPos;
DWORD           FX_ProcessEndPos;

DWORD           FX_CurSmpteHours;
DWORD           FX_CurSmpteMins;
DWORD           FX_CurSmpteSecs;
DWORD           FX_CurSmpteFrames;
DWORD           FX_CurSmpteSFrames;
DWORD           FX_CurSmpteStartOffsetHours;
DWORD           FX_CurSmpteStartOffsetMins;
DWORD           FX_CurSmpteStartOffsetSecs;
DWORD           FX_CurSmpteStartOffsetFrames;
DWORD           FX_CurSmpteStartOffsetSFrames;
DWORD           FX_CurSmpteFrameCount;
DWORD           FX_CurSmpteFormat;
DWORD           FX_CurSmpteMode;

HWND            FX_hWndSoundFile;
DWORD           FX_SFSampleRate;
DWORD           FX_SFResolution;
DWORD           FX_SFZoomX;
DWORD           FX_SFZoomY;
int             FX_SFZoomShiftY;
DWORD           FX_SFZoomXMagnify;
DWORD           FX_SFMarkBegPos;
DWORD           FX_SFMarkEndPos;
DWORD           FX_SFCursorSamplePos;
DWORD           FX_SFStartSamplePos;

DWORD           FX_SFWaveDBScale;
int             FX_SFWaveCenterLft;
int             FX_SFWaveCenterRgt;
int             FX_SFWaveDivider;
int             FX_SFWaveTop;
int             FX_SFWaveBot;
int             FX_SFWaveLft;
int             FX_SFWaveRgt;
DWORD           FX_SFWaveOkToDraw;

DWORD           FX_MTWaveDBScale;
int             FX_MTWaveCenterLft;
int             FX_MTWaveCenterRgt;
int             FX_MTWaveDivider;
```

```
    int             FX_MTWaveTop;
    int             FX_MTWaveBot;
    int             FX_MTWaveLft;
    int             FX_MTWaveRgt;
    int             FX_MTZoomShiftY;
    DWORD           FX_MTWaveTrack;
    DWORD           FX_MTCursorSamplePos;
    DWORD           FX_MTStartSamplePos;
    DWORD           FX_MTWaveOkToDraw;

    DWORD           FX_MTAutomationWriteFlag;
    DWORD           FX_XtraFlag2;
    DWORD           FX_XtraFlag3;
    DWORD           FX_XtraFlag4;
    DWORD           FX_XtraFlag5;
    DWORD           FX_XtraFlag6;
    DWORD           FX_XtraFlag7;
    DWORD           FX_XtraFlag8;
    DWORD           FX_XtraFlag9;
    DWORD           FX_XtraFlag10;
    DWORD           FX_XtraFlag11;
    DWORD           FX_XtraFlag12;

    DWORD           FX_CurTempoBeatsPerMin;
    DWORD           FX_CurTempoBeatsPerMeasure;
    DWORD           FX_CurTempoBeatRes;
    DWORD           FX_CurTempoTickRes;

    } IQS_SAWFXDATA;
```

**Next we define some equates. One of the important ones to mention here is the MAX_PARAM_SETS define. This determines how many different parameter sets your module will allow. This controls how many times your module can be patched. Each parameter set contains a complete set of control variables which it needs to operate. The data will be stored in arrays. If your module should only be patched once, then this value should be set to 1.**

```
//===========
// Constants
//===========
#define ALL_FILES                       1
#define PLG_FILE                        2

#define MAX_PARAM_SETS                  16
#define MAX_LIST_CHARS                  16

#define NONE_AVAILABLE                  0xffffffff
#define NO_ASSIGNMENT                   0xffffffff
#define NO_VALUE                        0xffffffff
#define END_OF_LIST                     0xffffffff

#define MAX_METER_INDEXES               512
#define MAX_METER_SEGMENTS              104
```

```
#define METER_RESPONSE_FACTOR_SLOW      2
#define METER_RESPONSE_FACTOR_MED       4
#define METER_RESPONSE_FACTOR_FAST      8
```

**These should be left intact and will be expanded as new functions are added to the API.**

```
// FX Misc Defines
//----------------
#define FX_ALL_PARAM_SETS               0xffffffff
#define FX_ALL_TRACKS                   0xffffffff
#define FX_ALL_CONTROLS                 0xffffffff

#define FX_WINDOW_DATA                  0x00000001
#define FX_SMPTE_DATA                   0x00000002
#define FX_TEMPO_DATA                   0x00000004

// FX Smpte Mode Definitions
//--------------------------
#define SMPTE_NOT_ACTIVE                0
#define SMPTE_GENERATE_ACTIVE           1
#define SMPTE_GENERATE_MTC_ACTIVE       2
#define SMPTE_GENERATE_BOTH_ACTIVE      3
#define SMPTE_TRIGGER_ACTIVE            4
#define SMPTE_TRIGGER_MTC_ACTIVE        5
#define SMPTE_CHASE_TRIGGER_ACTIVE      6
#define SMPTE_CHASE_TRIGGER_MTC_ACTIVE  7

// FX Smpte Format Defines
//------------------------
#define SMPTE_FORMAT_30N                1
#define SMPTE_FORMAT_30D                2
#define SMPTE_FORMAT_2997N              3
#define SMPTE_FORMAT_2997D              4
#define SMPTE_FORMAT_25                 5
#define SMPTE_FORMAT_24                 6
#define SMPTE_FORMAT_UNKNOWN            7

// FX Process Flag Definitions
//----------------------------
#define PROCESS_IDLE                    0
#define REALTIME_PLAYBACK               1
#define BUILD_TO_MIX_FILE               2
#define PROCESS_TO_SOUND_FILE           3
#define BUILD_TO_FX_MODULES             4
#define DATA_PRESCAN                    5

// FX Handler Function ID Definitions
//-----------------------------------
#define FX_SHUTDOWN                     1

#define FX_BEGIN_PROCESS                2
#define FX_END_PROCESS                  3
#define FX_BEGIN_XTRA                   4
```

```
#define FX_END_XTRA                         5

#define FX_OPEN_PLUGIN_WINDOW               6
#define FX_CLOSE_PLUGIN_WINDOW              7

#define FX_PATCH_NEW_PARAM_SET              8
#define FX_RESET_PARAM_SET                  9
#define FX_RECEIVE_PARAM_SET_DATA           10
#define FX_SEND_PARAM_SET_DATA              11
#define FX_INIT_PARAM_SET_DATA              12

#define FX_CHANGE_MT_RATE                   13
#define FX_CHANGE_MT_RES                    14
#define FX_CHANGE_MT_ACTIVE_TRACK           15
#define FX_CHANGE_MT_MARKS                  16
#define FX_CHANGE_SMPTE_INFO                17
#define FX_CHANGE_TEMPO_INFO                18

#define FX_CLEAR_CHASE_AUTOMATION           100
#define FX_STORE_CHASE_AUTOMATION           101
#define FX_DISPLAY_CHASE_AUTOMATION         102
#define FX_RESYNC_CHASE_AUTOMATION          103
#define FX_CHANGE_PROCESS_AUTOMATION        104
#define FX_DISPLAY_PROCESS_AUTOMATION       105

// FX Execute Function ID Definitions
//-------------------------------------
#define FORCE_FX_PRESCAN_STOP               1
#define FORCE_FX_PRESCAN_CONTINUE           2
#define MARK_MT_BEG                         3
#define MARK_MT_END                         4
#define CLEAR_MT_MARKS                      5
#define SPLIT_KEEP_MTENTRY                  6
#define SPLIT_MARK_MTENTRY                  7
#define SPLIT_REMOVE_MTENTRY                8
#define SPLIT_DELETE_MTENTRY                9
#define SELECT_MODE_ON                      10
#define SELECT_MODE_OFF                     11
#define EDL_UNDO_OFF                        12
#define EDL_UNDO_ON                         13
#define EDL_SAVE_UNDO                       14
#define EDL_RECALL_UNDO                     15

#define REFRESH_FX_DATA                     16

#define SET_MT_SAMPLE_POS                   17
#define SET_MT_SMPTE_FRAME_POS              18

#define START_MT_AUDIO_PLAY                 19
#define START_MT_AUDIO_REC                  20
#define START_MT_AUDIO_SRP                  21
#define STOP_MT_AUDIO                       22

#define SET_EDL_MODIFY_FLAG                 23
```

```
#define CHANGE_MT_ACTIVE_TRACK          24

#define SET_MAX_READ_AHEAD              25
#define SET_MIN_READ_AHEAD              26
#define REFRESH_HOST_DISPLAY            27

#define SET_SOLO_SWITCH_VALUE           50

#define SEND_TO_AUTOMATION              100
```

**These would be specific to your interface design.**

```
//=====================
// Control Definitions
//=====================

// Options Popup Menu
//-------------------
#define IDM_OPTIONS_LOAD_PRESETS        5000
#define IDM_OPTIONS_SAVE_PRESETS        5002
#define IDM_OPTIONS_ABOUT               5004

// Parameter List Menu
//--------------------
#define IDM_MENULIST_BASE               5100
```

**This module handles automation easily by defining each control type with a unique Control ID. The actual control values are stored in a double array defined by the MAX_PARAM_SETS and MAX_AUTO_CONTROLS equates.**

```
// Automation Control Id Values
//----------------------------
#define MAX_AUTO_CONTROLS               8

#define AUTO_BYPASS_SW                  0
#define AUTO_VOL_POT                    1

#define AUTO_VALUE_1                    2
#define AUTO_VALUE_2                    3
#define AUTO_VALUE_3                    4
#define AUTO_VALUE_4                    5
#define AUTO_VALUE_5                    6
#define AUTO_VALUE_6                    7
```

**Next come the procedure prototype definitions.**

```
//======================
// Procedure Prototypes
```

```
//=====================
```

**This must be defined for the dll module to operate in Windows.**

```
// Dll Main
//---------
BOOL            WINAPI      DllMain(HANDLE, DWORD, LPVOID);
```

**The next four routines must be defined and exported. The .def file in this module handles the export. These are the actual functions that interact directly with the host program.**

```
// Direct API Routines
//--------------------
void            __stdcall   FxInit(IQS_SAWFXDATA*);
void            __stdcall   FxMainHandler(DWORD);
void            __stdcall   FxProcessBuffer(DWORD, LPBYTE);
void            __stdcall   FxChangePosition(void);
```

**This group handles the individual functions requested of the plug-in by the host program.**

```
// Handler API Routines
//---------------------
void            __stdcall   PlugInShutDown(void);

void            __stdcall   PlugInBegProcess(void);
void            __stdcall   PlugInEndProcess(void);

void            __stdcall   PlugInOpenWindow(void);
void            __stdcall   PlugInCloseWindow(void);

void            __stdcall   PlugInPatchNewParamSet(void);
void            __stdcall   PlugInResetParamSet(void);
void            __stdcall   PlugInReceiveParamSetData(void);
void            __stdcall   PlugInSendParamSetData(void);
void            __stdcall   PlugInInitParamSetData(void);

void            __stdcall   PlugInChangeRate(void);
void            __stdcall   PlugInChangeRes(void);
void            __stdcall   PlugInChangeActiveTrack(void);
void            __stdcall   PlugInChangeMTMarks(void);
void            __stdcall   PlugInChangeSmpteInfo(void);
void            __stdcall   PlugInChangeTempoInfo(void);

void            __stdcall   PlugInClearChaseAutomation(void);
void            __stdcall   PlugInStoreChaseAutomation(void);
void            __stdcall   PlugInDisplayChaseAutomation(void);
void            __stdcall   PlugInReSyncChaseAutomation(void);
void            __stdcall   PlugInChangeProcessAutomation(void);
void            __stdcall   PlugInDisplayProcessAutomation(void);
```

**This group handles internal functions to initialize and shutdown the plug-in.**

```
// Internal Support Routines
//----------------------------
void            __stdcall   InitializeProc(void);
void            __stdcall   ShutdownProc(void);

void            __stdcall   InitThreads(void);
void            __stdcall   ShutDownThreads(void);
void            __stdcall   ForceThreadExit(HANDLE, DWORD, PDWORD);
DWORD           WINAPI      FxBackServiceCallback(LPVOID);
```

**This group handles internal functions specific to the plug-in's interface. The details of many of these routines will be skipped in this walk-through when they are not specific to the operation of the API.**

```
void            __stdcall   IQSGetModuleDirectory(LPSTR);
void            __stdcall   ExtractPath(LPSTR);
void            __stdcall   IQSGetCurDirectory(LPSTR);
void            __stdcall   AddBackSlashToPath(LPSTR);

void            __stdcall   CreateAboutWindow(void);
void            __stdcall   PaintAboutWindow(HDC);

LRESULT         CALLBACK    DLLWndMsgProc(HWND, UINT, WPARAM, LPARAM);

int             __stdcall   CheckBoundaries(LPRECT);

BOOL            __stdcall   PickOpenFileName(LPSTR, LPSTR, LPSTR, int, LPSTR);
BOOL            __stdcall   PickSaveFileName(LPSTR, LPSTR, LPSTR, int, LPSTR, DWORD);

void            __stdcall   RefreshBitmap(HDC, LPRECT, HBITMAP);
void            __stdcall   DisplayBitmapBtn(HDC, LPRECT, HBITMAP);

DWORD           __stdcall   ChkVKey(DWORD);
void            __stdcall   IQSMoveWindow(HWND);
void            __stdcall   GradientFillRect(HDC, LPRECT, int, int, int, int);

void            __stdcall   ClearAllMouseMsgs(void);
int             __stdcall   WaitMsecForEvent(int, UINT);
```

**This group handles plug-in control routines for the interface. The details of many of these routines will be skipped in this walk-through when they are not specific to the operation of the API.**

```
// PlugIn Control Routines
```

```
//------------------------
void            __stdcall   CreatePlugInWindow(DWORD, HWND);
void            __stdcall   DestroyPlugInWindow(DWORD);

LRESULT         CALLBACK    DLLWndPlugInProc(HWND, UINT, WPARAM, LPARAM);

void            __stdcall   DisplayPlugInValue(HDC, DWORD, int, DWORD);
void            __stdcall   PresetPlugInBtns(DWORD);

void            __stdcall   CheckPlugInZones(void);
DWORD           __stdcall   CheckPlugInZonesRight(void);

void            __stdcall   DrawVertKnob(HDC, LPRECT, DWORD, DWORD);
void            __stdcall   LockToVertKnob(void);
void            __stdcall   MoveVertKnob(void);
void            __stdcall   SetVertKnobValue(void);
void            __stdcall   SetVertKnobValueDef(void);

void            __stdcall   PlusMinusCheck(void);
void            __stdcall   PlusMinusEnd(void);
void            __stdcall   ActionControlScroll(void);
void            __stdcall   MoveControlScroll(void);
void            __stdcall   ActionControlMenuList(void);
void            __stdcall   ActionSetToDefaultValue(void);
void            __stdcall   ActionUpdateControl(void);

void            __stdcall   SendAutomation(DWORD, int, int, int);

void            __stdcall   CreateOptionsSelectionMenu(void);

void            __stdcall   SavePlugInPreset(void);
void            __stdcall   LoadPlugInPreset(void);
```

**This group does the actual work of processing the data buffers passed from the host program.**

```
void            __stdcall   InitPlugInVariables(DWORD, DWORD);
void            __stdcall   InitPlugInMeter(DWORD);
void            __stdcall   ResetParamSet(DWORD);

void            __stdcall   DoPlugInProcess16(void);
void            __stdcall   DoPlugInProcess32(void);

void            __stdcall   ResetMeterDisplay(DWORD);
void            __stdcall   DisplayAllMeters(void);
```

**Next come the Global variable definitions. Though many programmers have adopted the belief that global variables should not be used, I find that with a little care, they can increase performance significantly compared to passing numerous parameters from routine to routine.**

```
//==================
// Global Variables
```

```
//==================

// General Variables
//------------------
```

**This variable defines the plug-in name, which is passed back to the host program during the FxInit function.**

```
char                            szFxTitle[] = "FX API TestPlugIn";
```

**This variable becomes the main pointer to the shared memory data structure which forms the basis of all communication between the host program and the plug-in. This pointer is passed to the plug-in during the FxInit function.**

```
IQS_SAWFXDATA*                  FxPtrSawData = NULL;
```

**This next group of variables represent values that get used numerous times in numerous routines, and storing their values once can make the code easier to follow and more efficient.**

```
DWORD                           FxProgramVersion = 0;
DWORD                           FxPlugInIndex = 0;
DWORD                           FxMaxBufferSize = 0;

DWORD                           FxSampleRate;
DWORD                           FxResolution;

DWORD                           FxReturnTrkOffset;
DWORD                           FxOutputTrkOffset;

DWORD                           FxProcessFlag = 0;
DWORD                           FxPreScanFlag = 0;

DWORD                           FxProcessParamSet = 0;
DWORD                           FxProcessBytesPerSample = 4;
LPBYTE                          FxProcessBuffPtr;
DWORD                           FxProcessBuffSize;
DWORD                           FxProcessPos;

DWORD                           FxSamplePos;

DWORD                           FxActiveParamSet = 0;

HINSTANCE                       hInstFx;
HWND                            hWndSawMain;
HFONT                           hIQSSysFont;
```

**This next group of variables is involved with internal interface control.**

```
int                             ScreenMaxH;
int                             ScreenMaxV;
int                             CurMouseX = 0;
int                           CurMouseY = 0;
int                           OldMouseX = 0;
int                           OldMouseY = 0;

char                          szWorkBuff[512];
char                          szStringBuff[512];

DWORD                         MemoryAllocFailureFlag = 0;
DWORD                         BuffWorkSize = 0x00020000;
LPBYTE                        PtrBuffWork1;

// File Variables
//---------------
OFSTRUCT                      ofFileStructure;
OPENFILENAME                  ofFileName;

char                          szOrgPath[MAX_PATH];
char                          szOpenName[MAX_PATH];
char                          szSaveName[MAX_PATH];
char                          szWork1Name[MAX_PATH];
char                          szDefaultPath[MAX_PATH];

char                          szAllFilters[] = "All Files  *.*\0*.*\0"
                                               "PlugIn Preset File  *.plg\0*.plg\0"
                                               "\0";

char                          szAllQual[] = ".*";
char                          szPlgQual[] = ".plg";

char                          szAppendExt[] = "\0\0\0\0\0\0";

char                          LastPlugInFileName[256];

// Knob Variables
//---------------
DWORD                         VertKnobActive = 0;
DWORD                         VertKnobCtrlId = NO_VALUE;
int                           VertKnobPos = 0;
int                           VertKnobPosDef = 0;
HWND                          VertKnobWindow;
HPEN                          VertKnobPen = 0;
HBITMAP                       VertKnobBackBitmap;
RECT                          VertKnobRect;

// Special Cursors
//----------------
HCURSOR                       hPlusMinusCursor = 0;
char                          szPlugInPlusMinusCursorName[] = "Plus_Minus_Cursor";

// Action Control Variables
//-------------------------
PROC                          ActionControlFunction;

int                           PlusMinusActive = 0;
int                         PlusMinusCenter = 0;

DWORD                       ActiveControlType;
int                         ActiveControlValue;
int                         ActiveControlValueMin;
int                         ActiveControlValueMax;
int                         ActiveControlValueDef;
int                         ActiveControlMultiplier;
RECT                        ActiveControlRect;
LPSTR                       ActiveControlListPtr;
```

**These variables are used for the plug-in function requests of the host program.**

```
// Communications Messages To SAW
//-------------------------------
char                        szSawFxExecuteFunctionMsg[] = "SawFxExecuteFunction";
DWORD                       SawFxExecuteFunctionMsg = 0;
```

**More variables involved with internal interface control.**

```
// Class Names
//------------
char                        szPlugInClass[] = "FX_PLUGIN";
char                        szMsgClass[] = "FX_PLUGIN_MSG";

// Bitmaps
//--------
char                        szPlugInMainName[] = "PlugIn_Main_Bit";
char                        szPlugInBypassBtnName[] = "PlugIn_Bypass_Btn_Bit";
char                        szPlugInMeterName[] = "PlugIn_Meter_Bit";

HBITMAP                     hAboutBitmap = 0;
BITMAP                      AboutBitMapHdr;

HBITMAP                     hPlugInMainBitmap = 0;
BITMAP                      PlugInMainBitmapHdr;

HBITMAP                     hPlugInBypassBtnBitmap = 0;
BITMAP                      PlugInBypassBtnBitmapHdr;

HBITMAP                     hPlugInMeterBitmap = 0;
BITMAP                      PlugInMeterBitmapHdr;

// Drawing Tools
//--------------
HBRUSH                      hBlackBrush;
HBRUSH                      hGrayBrush;

HPEN                        hVertKnobPen;

COLORREF                    hBlackColor;
COLORREF                    hTextColor;
```

```
// Selection Menus
//----------------
HMENU                       hMenuOptions;
HMENU                       hMenuList;

// About Message Window
//---------------------
HWND                        hWndAbout = 0;
DWORD                       AboutModalFlag = 0;

// Thread Variables
//-----------------
HANDLE                      hFxBackThread;
DWORD                       FxBackThreadID;
DWORD                       FxBackThreadExitFlag = 0;
DWORD                       FxBackThreadSuspendFlag = 0;
DWORD                       FxBackThreadSleepTime = 80;
```

**These variables are involved with the plug-in window displays and parameter set data control.**

```
// PlugIn Window
//--------------
HWND                        hWndPlugIn[MAX_PARAM_SETS];
HDC                         PlugInDC[MAX_PARAM_SETS];

int                         PlugInXStart[MAX_PARAM_SETS];
int                         PlugInYStart[MAX_PARAM_SETS];
int                         PlugInWidth[MAX_PARAM_SETS];
int                         PlugInHeight[MAX_PARAM_SETS];

DWORD                       PlugInBytesPerSample[MAX_PARAM_SETS];
DWORD                       PlugInTrack[MAX_PARAM_SETS];

DWORD                       PlugInParamSetCount;
DWORD                       PlugInParamSetTable[MAX_PARAM_SETS];
```

**These variables are involved with the plug-in meter display.**

```
DWORD                       PlugInMeterIndexHead[MAX_PARAM_SETS];
DWORD                       PlugInMeterIndexTail[MAX_PARAM_SETS];

int                         PlugInMeterLastValueL[MAX_PARAM_SETS];
int                         PlugInMeterLastValueR[MAX_PARAM_SETS];

DWORD                       PlugInMeterPos[MAX_PARAM_SETS][MAX_METER_INDEXES];
int                         PlugInMeterValueL[MAX_PARAM_SETS][MAX_METER_INDEXES];
int                         PlugInMeterValueR[MAX_PARAM_SETS][MAX_METER_INDEXES];

// Meter Factor Table
//-------------------
int                         MeterTable[MAX_METER_SEGMENTS] = {
```

```
                            // Mute
                            //-----
                            0, 8747, 10194, 11642, 13090,
                            14538, 15986, 17434, 20321, 23206,
                            26091, 28976, 31861, 34746, 40499,
                            46249, 51999, 57749, 63499, 69249,
                            80708, 92169, 103630, 115091, 126552,
                            138013, 159475, 180936, 202397, 223858,
                            245319, 266780, 310932, 355084, 399236,
                            443388, 487540, 531692, 619688, 707683,
                            795678, 883673, 971668, 1059663, 1098594,
                            1137521, 1176448, 1215375, 1254302, 1293229,

                            // -16db
                            //------
                            1332156, 1384147, 1436130, 1488113, 1540096,
                            1592079, 1644062, 1696045, 1748028, 1800011,
                            1851994, 1903977, 1955960, 2007943, 2059926,
                            2111909, 2193939, 2275963, 2357987, 2440011,
                            2522035, 2604059, 2686083, 2768107, 2850131,
                            2932155, 3014179, 3096203, 3178227, 3260251,

                            // -8db
                            //-----
                            3342275, 3486735, 3631195, 3775655, 3920115,
                            4064575, 4209035, 4454895, 4700752, 4946609,
                            5192466, 5438323, 5684180, 5930037, 6175894,
                            6421751, 6667608, 6913465, 7159322, 7405179,
                            7651036, 7896893, 8142750, 8388607,
                            };
```

**These variables define the plug-in parameter control label displays and values.**

```
char                        PlugInParamLabel[6][MAX_LIST_CHARS] =
                            {
                            "Output Format",
                            "Meter Response",
                            "Parameter 3",
                            "Parameter 4",
                            "Parameter 5",
                            "Parameter 6",
                            };

char                        OutputFormatTable[6][MAX_LIST_CHARS] =
                            {
                            "Stereo",
                            "Mono",
                            "Lft Only",
                            "Rgt Only",
                            "Lft Mono",
                            "Rgt Mono",
                            };
```

```
char                        MeterResponseTable[3][MAX_LIST_CHARS] =
                            {
                            "Slow",
                            "Med",
                            "Fast",
                            };

int                         MeterResponseValue[3] =
                            {
                            METER_RESPONSE_FACTOR_SLOW,
                            METER_RESPONSE_FACTOR_MED,
                            METER_RESPONSE_FACTOR_FAST,
                            };
```

**These variables define the plug-in rectangle action and display zones of the interface.**

```
RECT                        RectPlugInOptions = {339, 83, 391, 110};
RECT                        RectPlugInCloseBtn = {369, 11, 393, 20};
RECT                        RectPlugInBypassBtn = {342, 36, 385, 58};
RECT                        RectPlugInBypassDisplayBtn = {342, 36, 393, 64};
RECT                        RectPlugInTrack = {335, 167, 391, 187};

RECT                        RectPlugInVolPot = {294, 40, 306, 150};

RECT                        RectPlugInValue1 = {13, 12, 101, 33};
RECT                        RectPlugInValue2 = {13, 42, 101, 63};
RECT                        RectPlugInValue3 = {13, 73, 101, 94};
RECT                        RectPlugInValue4 = {13, 104, 101, 125};
RECT                        RectPlugInValue5 = {13, 135, 101, 156};
RECT                        RectPlugInValue6 = {13, 166, 101, 187};

RECT                        RectPlugInLabel1 = {113, 13, 260, 33};
RECT                        RectPlugInLabel2 = {113, 43, 260, 63};
RECT                        RectPlugInLabel3 = {113, 74, 260, 94};
RECT                        RectPlugInLabel4 = {113, 104, 260, 125};
RECT                        RectPlugInLabel5 = {113, 136, 260, 156};
RECT                        RectPlugInLabel6 = {113, 167, 260, 187};

RECT                        RectPlugInMeterL = {314, 49, 315, 152};
RECT                        RectPlugInMeterR = {317, 49, 318, 152};
RECT                        RectPlugInMeterPeakL = {314, 37, 315, 45};
RECT                        RectPlugInMeterPeakR = {317, 37, 318, 45};
```

**These variables are the plug-in data arrays for each control of each parameter set.**

```
// Data Storage Arrays
//--------------------
int                         PlugInDataValue[MAX_PARAM_SETS][MAX_AUTO_CONTROLS];
int                         PlugInDefaultDataValue[MAX_PARAM_SETS][MAX_AUTO_CONTROLS];
int                         PlugInAutoDataValue[MAX_PARAM_SETS][MAX_AUTO_CONTROLS];
```

**These variables are used in performing the plug-in process.**

```
// Parameter Set Special Variables
//--------------------------------
int                         FxVolFactor[MAX_PARAM_SETS];
DWORD                       FxOutputFormat[MAX_PARAM_SETS];
int                         PlugInMeterDecayFactor[MAX_PARAM_SETS];
```

**These variables are misc text string messages.**

```
// Text Message String Variables
//------------------------------
char                        AboutMsg[] =                    "Studio Fx API TestPlugIn\n"
                                                            "Version 1.0\n"
                                                            "From\n"
                                                            "Innovative Quality Software\n\n"
                                                            "(c) Copyright 2001 - Bob Lentini\n";

char                        WarningMsg[] =                  "WARNING!";
char                        ImproperPresetMsg[] =           "Improper Preset Format!";
char                        CorruptedErrorMsg[] =           "Corrupted Data Settings For This PlugIn\nPlease Reset Parameters!";

char                        MaximumAssignmentMsg[] =        "This Effect Has Reached Its Maximum Assignments!";

//----
```

## FX_API_TestPlugIn.cpp file.

**This file contains the actual plug-in code.**

```
//=============================================
// SAWStudio FX API TestPlugIn - DLL Routines
//
// copyright 2001 By Bob Lentini
//=============================================

#include <windows.h>
#include "FX_API_TestPlugIn.h"
```

**This routine is required by Windows for all dll applications. During the DLL_PROCESS_ATTACH call we simply save the dll instance handle for use in other function calls.**

```
//=========
// DllMain
//=========
BOOL    WINAPI      DllMain(HANDLE hDLL, DWORD dwReason, LPVOID lpReserved)

{
switch(dwReason)
```

```
    {
    case DLL_PROCESS_ATTACH:

        // Save Instance Handle For Later Use
        //------------------------------------
        hInstFx = (HINSTANCE)hDLL;

        break;

    case DLL_THREAD_ATTACH:
        break;

    case DLL_THREAD_DETACH:
        break;

    case DLL_PROCESS_DETACH:
        break;
    }

return(TRUE);
}
```

**This routine is the first API function to be called when the host program initializes this dll module. The first thing done is to save the shared memory structure pointer for later use. It also saves several other important variables referenced in that structure for later use. The plug-in then calls its own internal init routine to setup everything it needs to function properly and resets all of its parameter sets to default values. It then clears the process variables and send the plug-in name back to the host program.**

```
//=======================
// SAWStudio API Routine
//
// FX Initialization
//=======================

void    __stdcall   FxInit(IQS_SAWFXDATA* SawFxDataPtr)

{
// Save Shared Memory Pointer
//----------------------------
FxPtrSawData = SawFxDataPtr;

// Take Data From Saw
//--------------------
FxPlugInIndex = FxPtrSawData->FX_PlugInIndex;
hWndSawMain = FxPtrSawData->FX_hWndMain;
hIQSSysFont = FxPtrSawData->FX_IQSSysFont;

FxProgramVersion = FxPtrSawData->FX_VersionNum;
FxMaxBufferSize = FxPtrSawData->FX_MaxBufferByteSize;

FxSampleRate = FxPtrSawData->FX_MTSampleRate;
FxResolution = FxPtrSawData->FX_MTResolution;
FxReturnTrkOffset = FxPtrSawData->FX_MultiTrackRtnTrackOffset;
FxOutputTrkOffset = FxPtrSawData->FX_MultiTrackOutTrackOffset;

// Initialize PlugIn
//------------------
InitializeProc();

// Reset All Parameter Sets
//-------------------------
ResetParamSet(FX_ALL_PARAM_SETS);

// Clear Process Variables
//------------------------
FxProcessParamSet = 0;
FxProcessFlag = PROCESS_IDLE;
FxPreScanFlag = PROCESS_IDLE;

// Send Back Title String
//-----------------------
lstrcpy(FxPtrSawData->FX_Title, szFxTitle);
return;
}
```

**This routine handles all of the host program requests. It is a simple conditional parsing routine that looks at the function ID parameter and calls the proper function.**

```
//======================
// SAWStudio API Routine
//
// FX Main Handler
//======================

void  __stdcall  FxMainHandler(DWORD TempFunctionID)

{
// Call The Proper Routine Based On The Function ID
//-----------------------------------------------
if(TempFunctionID == FX_SHUTDOWN)
    {
    PlugInShutDown();
    return;
    }

if(TempFunctionID == FX_BEGIN_PROCESS)
    {
    PlugInBegProcess();
    return;
    }

if(TempFunctionID == FX_END_PROCESS)
    {
    PlugInEndProcess();
    return;
    }

if(TempFunctionID == FX_BEGIN_XTRA)
    {
    return;
    }

if(TempFunctionID == FX_END_XTRA)
    {
    return;
    }

if(TempFunctionID == FX_OPEN_PLUGIN_WINDOW)
    {
    PlugInOpenWindow();
    return;
    }

if(TempFunctionID == FX_CLOSE_PLUGIN_WINDOW)
    {
    PlugInCloseWindow();
    return;
    }

if(TempFunctionID == FX_PATCH_NEW_PARAM_SET)
    {
    PlugInPatchNewParamSet();
    return;
    }

if(TempFunctionID == FX_RESET_PARAM_SET)
    {
    PlugInResetParamSet();
    return;
    }

if(TempFunctionID == FX_RECEIVE_PARAM_SET_DATA)
    {
    PlugInReceiveParamSetData();
    return;
    }

if(TempFunctionID == FX_SEND_PARAM_SET_DATA)
    {
    PlugInSendParamSetData();
    return;
    }

if(TempFunctionID == FX_INIT_PARAM_SET_DATA)
    {
    PlugInInitParamSetData();
    return;
    }
```

```
if(TempFunctionID == FX_CHANGE_MT_RATE)
    {
    PlugInChangeRate();
    return;
    }

if(TempFunctionID == FX_CHANGE_MT_RES)
    {
    PlugInChangeRes();
    return;
    }

if(TempFunctionID == FX_CHANGE_MT_ACTIVE_TRACK)
    {
    PlugInChangeActiveTrack();
    return;
    }

if(TempFunctionID == FX_CHANGE_MT_MARKS)
    {
    PlugInChangeMTMarks();
    return;
    }

if(TempFunctionID == FX_CHANGE_SMPTE_INFO)
    {
    PlugInChangeSmpteInfo();
    return;
    }

if(TempFunctionID == FX_CHANGE_TEMPO_INFO)
    {
    PlugInChangeTempoInfo();
    return;
    }

if(TempFunctionID == FX_CLEAR_CHASE_AUTOMATION)
    {
    PlugInClearChaseAutomation();
    return;
    }

if(TempFunctionID == FX_STORE_CHASE_AUTOMATION)
    {
    PlugInStoreChaseAutomation();
    return;
    }

if(TempFunctionID == FX_DISPLAY_CHASE_AUTOMATION)
    {
    PlugInDisplayChaseAutomation();
    return;
    }
```

```
if(TempFunctionID == FX_RESYNC_CHASE_AUTOMATION)
    {
    PlugInReSyncChaseAutomation();
    return;
    }

if(TempFunctionID == FX_CHANGE_PROCESS_AUTOMATION)
    {
    PlugInChangeProcessAutomation();
    return;
    }

if(TempFunctionID == FX_DISPLAY_PROCESS_AUTOMATION)
    {
    PlugInDisplayProcessAutomation();
    return;
    }

return;
}
```

**This routine is called directly by the host program for every buffer of data that requires processing through the plug-in. This routine does the actual processing on the data stream. I recommend writing this routine in assembly language, if you can, to provide the most efficiency during the multitrack loop. This routine gets the current parameters for this call and decides whether it needs to process. It then calls the proper processing routine based on the buffer format. The final buffer size is passed back to the host program to be passed on to the next plug-in in line. Remember not to alter the buffer size beyond the allowable FxMaxBufferSize. You may withhold the complete buffer, if needed, by copying it into an internal allocated work buffer and return zero bytes back, gathering multiple buffers of data before returning processed data. The host program will maintain sync with other tracks automatically.**

```
//======================
// SAWStudio API Routine
//
// FX Process Buffer
//======================

void  __stdcall  FxProcessBuffer(DWORD ParamSet, LPBYTE BuffPtr)

{
// Set Process Variables For This Call
//-----------------------------------
FxProcessParamSet = ParamSet;
FxProcessBuffPtr = BuffPtr;
FxProcessBuffSize = FxPtrSawData->FX_BufferByteSize;
FxProcessBytesPerSample = FxPtrSawData->FX_BufferBytesPerSample;
FxProcessPos = FxPtrSawData->FX_CurProcessPos;

// This Plug-In Only Processes Stereo Buffers
//------------------------------------------
if(FxPtrSawData->FX_BufferChans != 2)
    return;

// Skip Process If PlugIn Is Bypassed
//-------------------------------------
if(PlugInDataValue[FxProcessParamSet][AUTO_BYPASS_SW])
    return;

// Process The Correct Buffer Format
//------------------------------------
if(FxProcessBytesPerSample == 4)
    DoPlugInProcess32();
else
    DoPlugInProcess16();

// Set Final BufferSize If Your Process Changes It
// Note That The Size Can Not Grow Beyond FxMaxBufferSize
//------------------------------------------------------
FxPtrSawData->FX_BufferByteSize = FxProcessBuffSize;

return;
}
```

**This routine is called in a Time Critical Thread to pass the current playback/record sample position to each patched in plug-in. This call may be ignored by simply returning immediately. If your plug-in is displaying meter information, or other display info that must be in sync with the audio being heard, then use this function to display the data. It is recommended that the display data be calculated during the FxProcessBuffer function and the results stored with the process position referenced in the storage. During this function, compare the current sample position with the next referenced process position and display the data at the proper time. This function should be performed as fast as possible to avoid pulling the system down during this thread execution.**

```
//=======================
// SAWStudio API Routine
//
// FX Change Position
//=======================

void  __stdcall  FxChangePosition(void)

{
// Get Parameter Data
//-------------------
FxSamplePos = FxPtrSawData->FX_CurSamplePos;

// Display All Meters At Current Sample Position Reference
//--------------------------------------------------------
DisplayAllMeters();
return;
}
```

**This is one of the Main Handler routines. This is called when the host program is about to be shutdown. This routine should close down the plug-in, clean up memory allocations, shutdown all threads, and close all open windows.**

```
//===========================
// SAWStudio Handler Routine
//
```

```
// PlugIn Shutdown
//===========================

void    __stdcall   PlugInShutDown(void)

{
// ShutDown PlugIn
//----------------
ShutdownProc();

return;
}
```

**This is one of the Main Handler routines. This is called when the host program is about to begin a process operation. This function gives the plug-in the chance to initialize all needed variables and get ready to process data.**

```
//===========================
// SAWStudio Handler Routine
//
// PlugIn Begin Process
//===========================

void    __stdcall   PlugInBegProcess(void)

{
// Set Process Variables For This Call
//-----------------------------------
FxProcessParamSet = FxPtrSawData->FX_Handler_DWord_Param1;
FxProcessBytesPerSample = FxPtrSawData->FX_Handler_DWord_Param2;
FxProcessFlag = FxPtrSawData->FX_ProcessFlag;

if(FxProcessFlag == DATA_PRESCAN)
    FxPreScanFlag = 1;
else
    FxPreScanFlag = 0;

// Init This Parameter Set For Processing
//--------------------------------------
InitPlugInVariables(FxProcessParamSet, FX_ALL_CONTROLS);
InitPlugInMeter(FxProcessParamSet);

return;
}
```

**This is one of the Main Handler routines. This is called at the end of a process operation and gives the plug-in a chance to clean up after processing is finished.**

```
//===========================
// SAWStudio Handler Routine
//
// PlugIn End Process
```

```
//============================

void    __stdcall   PlugInEndProcess(void)

{
// Reset Process Variables
//-------------------------
FxProcessParamSet = FxPtrSawData->FX_Handler_DWord_Param1;

FxProcessFlag = PROCESS_IDLE;
FxPreScanFlag = PROCESS_IDLE;

// CleanUp Parameter Set After Processing If Needed
//-------------------------------------------------
ResetMeterDisplay(FxProcessParamSet);

return;
}
```

**This is one of the Main Handler routines. This is called when the host program needs to open the interface window for the plug-in. The plug-in gathers the parameter data, and checks the requested parameter set to see that it has been marked as patched. If not, it collects more needed information and patches it now, and marks it as such. In either case, it then opens its display window.**

```
//============================
// SAWStudio Handler Routine
//
// PlugIn Open Window
//============================

void    __stdcall   PlugInOpenWindow(void)

{
DWORD   TempParamSet;
HWND    TempRefWindow;

// Get Parameter Data
//--------------------
TempParamSet = FxPtrSawData->FX_Handler_DWord_Param1;
TempRefWindow = (HWND)FxPtrSawData->FX_Handler_DWord_Param3;

if(PlugInParamSetTable[TempParamSet] == 0)
    {
    // Missing Patch... Add It
    //-------------------------
    MessageBox(NULL, CorruptedErrorMsg, WarningMsg, MB_TASKMODAL | MB_TOPMOST | MB_ICONEXCLAMATION | MB_OK);

    // Save Track Information
    //------------------------
    PlugInTrack[TempParamSet] = FxPtrSawData->FX_Handler_DWord_Param2;

    // Save Format Information
    //-------------------------
    PlugInBytesPerSample[TempParamSet] = FxPtrSawData->FX_Handler_DWord_Param4;

    // Init PlugIn Variables If Needed
    //--------------------------------
    InitPlugInVariables(TempParamSet, FX_ALL_CONTROLS);

    // Mark This Set Active In Table
    //----------------------------
    PlugInParamSetTable[TempParamSet] = 1;

    // Increment Active Set Count
    //-------------------------
    PlugInParamSetCount++;
    }

// Open PlugIn Window
//-------------------
CreatePlugInWindow(TempParamSet, TempRefWindow);
return;
}
```

**This is one of the Main Handler routines. This is called when the host program needs the plug-in window to be closed. This does not un-patch the plug-in from the loop, just hides its display interface.**

```
//===========================
// SAWStudio Handler Routine
//
// PlugIn Close Window
//===========================

void    __stdcall   PlugInCloseWindow(void)

{
DWORD   TempParamSet;

// Get Parameter Data
//-------------------
TempParamSet = FxPtrSawData->FX_Handler_DWord_Param1;

// Close PlugIn Window
//--------------------
DestroyPlugInWindow(TempParamSet);
return;
}
```

**This is one of the Main Handler routines. This is called when the host program is patching in a new instance of the plug-in. The plug-in should search its internal array table for an empty parameter set and patch the new plug-in there. It should return that parameter set value back to the host. The plug-in also saves important track and format information and initializes its variables for that parameter set. If the count has reached the MAX_PARAM_SET value, then the plug-in does not patch and returns the  NONE_AVAILABLE value back to the host.**

```
//================================
// SAWStudio Handler Routine
//
// PlugIn Patch New Parameter Set
//==================================

void    __stdcall   PlugInPatchNewParamSet(void)

{
DWORD   TempParamSet;

// Are There Any Available Parameter Sets?
//------------------------------------------
if(PlugInParamSetCount < MAX_PARAM_SETS)
    {
    // Search For The First Available Set
    //-------------------------------------
    for(TempParamSet = 0; TempParamSet < MAX_PARAM_SETS; TempParamSet++)
        {
        if(PlugInParamSetTable[TempParamSet] == 0)
            {
            // Save Format Information
            //-------------------------
            PlugInBytesPerSample[TempParamSet] = FxPtrSawData->FX_Handler_DWord_Param1;

            // Save Track Information
            //------------------------
            PlugInTrack[TempParamSet] = FxPtrSawData->FX_Handler_DWord_Param2;

            // Init PlugIn Variables If Needed
            //---------------------------------
            InitPlugInVariables(TempParamSet, FX_ALL_CONTROLS);

            // Pass New Parameter Set Value Back To Saw
            //------------------------------------------
            FxPtrSawData->FX_ParamSet = TempParamSet;

            // Mark This Set Active In Table
            //-------------------------------
            PlugInParamSetTable[TempParamSet] = 1;

            // Increment Active Set Count
            //----------------------------
            PlugInParamSetCount++;
            return;
            }
        }
    }

// None Available
//----------------
MessageBox(NULL, MaximumAssignmentMsg, WarningMsg, MB_TASKMODAL | MB_TOPMOST | MB_ICONEXCLAMATION | MB_OK);

FxPtrSawData->FX_ParamSet = NONE_AVAILABLE;
return;
}
```

**This is one of the Main Handler routines. This is called when a plug-in is un-patched. The plug-in should reset this parameter set, clear it from its internal array table and close any open window interface.**

```
//============================
// SAWStudio Handler Routine
//
// PlugIn Reset Parameter Set
//============================

void    __stdcall   PlugInResetParamSet(void)

{
DWORD   TempParamSet;

// Get Parameter Data
//-------------------
TempParamSet = FxPtrSawData->FX_Handler_DWord_Param1;

// Reset Parameter Set
//--------------------
ResetParamSet(TempParamSet);

// If The Window Exists... Close It
//---------------------------------
if(hWndPlugIn[TempParamSet])
    DestroyPlugInWindow(TempParamSet);

return;
}
```

**This is one of the Main Handler routines. This is called when the host program needs to send previously saved parameter data to the plug-in, for example when a session file is first opened.  Notice that the code first checks the 32 byte header string at the front of the buffer for a match. If this data is not the correct format for this plug-in, the code sets an error flag and returns immediately to the host. The parameter set value could be a call to receive just one specific parameter set or all parameter sets. The code handles both situations by first checking the parameter set value and adjusting the loop accordingly. The data is read and stored to each designated parameter set. Notice that control data is stored into the default data array as well as copied to the current data array. The plug-in variables are then initialized, the patched parameter set table is updated and the patched counter is incremented. The display position and visibility data is read, and if the plug-in was saved in a visible state, the plug-in window is opened. If everything goes correctly, a zero value is returned to the host.**

```
//===================================
// SAWStudio Handler Routine
//
// PlugIn Receive Parameter Set Data
//===================================

void    __stdcall   PlugInReceiveParamSetData(void)

{
```

```
DWORD   TempParamSet;
DWORD   TempBuffSize;
LPBYTE  TempBuffPtr;
DWORD   TempOffset;
DWORD   TempCount;
DWORD   TempMaxCount;
DWORD   TempSet;
DWORD   TempVisibleFlag;

// Get Parameter Data
//-------------------
TempParamSet = FxPtrSawData->FX_Handler_DWord_Param1;
TempBuffSize = FxPtrSawData->FX_Handler_DWord_Param2;
TempBuffPtr = FxPtrSawData->FX_Handler_Ptr_Param1;

// Check Header For A Match
//------------------------
if(lstrcmp((LPSTR)TempBuffPtr, "FX API TEST PLUGIN PRESETS 001 ") != 0)
    {
    // Not Proper Format
    //------------------
    FxPtrSawData->FX_Handler_DWord_Param3 = 1;
    return;
    }

// Offset Past The Header
//----------------------
TempOffset = 32;

// Reset The Parameter Set(s) (Might Be All Sets)
//----------------------------------------------
ResetParamSet(TempParamSet);

// Set Max Count
//-------------
if(TempParamSet == FX_ALL_PARAM_SETS)
    TempMaxCount = MAX_PARAM_SETS;
else
    TempMaxCount = 1;

// Read Individual Settings For Each Active ParamSet
//-------------------------------------------------
for(TempCount = 0; TempCount <= TempMaxCount; TempCount++)
    {
    // Get Parameter Set Value
    //-----------------------
    TempSet = *(PDWORD)(TempBuffPtr + TempOffset);
    TempOffset = TempOffset + 4;

    // End Of List... Exit Loop
    //------------------------
    if(TempSet == END_OF_LIST)
        break;
```

```
// Read PlugIn Data For This Set
//------------------------------
PlugInTrack[TempSet] = *(PDWORD)(TempBuffPtr + TempOffset);
TempOffset = TempOffset + 4;

PlugInBytesPerSample[TempSet] = *(PDWORD)(TempBuffPtr + TempOffset);
TempOffset = TempOffset + 4;

PlugInDefaultDataValue[TempSet][AUTO_BYPASS_SW] = *(PDWORD)(TempBuffPtr + TempOffset);
TempOffset = TempOffset + 4;

PlugInDefaultDataValue[TempSet][AUTO_VOL_POT] = *(PDWORD)(TempBuffPtr + TempOffset);
TempOffset = TempOffset + 4;

PlugInDefaultDataValue[TempSet][AUTO_VALUE_1] = *(PDWORD)(TempBuffPtr + TempOffset);
TempOffset = TempOffset + 4;

PlugInDefaultDataValue[TempSet][AUTO_VALUE_2] = *(PDWORD)(TempBuffPtr + TempOffset);
TempOffset = TempOffset + 4;

PlugInDefaultDataValue[TempSet][AUTO_VALUE_3] = *(PDWORD)(TempBuffPtr + TempOffset);
TempOffset = TempOffset + 4;

PlugInDefaultDataValue[TempSet][AUTO_VALUE_4] = *(PDWORD)(TempBuffPtr + TempOffset);
TempOffset = TempOffset + 4;

PlugInDefaultDataValue[TempSet][AUTO_VALUE_5] = *(PDWORD)(TempBuffPtr + TempOffset);
TempOffset = TempOffset + 4;

PlugInDefaultDataValue[TempSet][AUTO_VALUE_6] = *(PDWORD)(TempBuffPtr + TempOffset);
TempOffset = TempOffset + 4;

// Set All Current Values To New Defaults
//---------------------------------------
CopyMemory(&PlugInDataValue[TempSet][0], &PlugInDefaultDataValue[TempSet][0], MAX_AUTO_CONTROLS * 4);

// Init PlugIn Variables
//----------------------
InitPlugInVariables(TempSet, FX_ALL_CONTROLS);

// Mark This Set Active In Table
//------------------------------
PlugInParamSetTable[TempSet] = 1;

// Increment Active Set Count
//---------------------------
PlugInParamSetCount++;

// Read Window Info
//-----------------
PlugInXStart[TempSet] = *(PDWORD)(TempBuffPtr + TempOffset);
TempOffset = TempOffset + 4;

PlugInYStart[TempSet] = *(PDWORD)(TempBuffPtr + TempOffset);
```

```
    TempOffset = TempOffset + 4;

    PlugInWidth[TempSet] = *(PDWORD)(TempBuffPtr + TempOffset);
    TempOffset = TempOffset + 4;

    PlugInHeight[TempSet] = *(PDWORD)(TempBuffPtr + TempOffset);
    TempOffset = TempOffset + 4;

    TempVisibleFlag = *(PDWORD)(TempBuffPtr + TempOffset);
    TempOffset = TempOffset + 4;

    // Display Window If Needed
    //-------------------------
    if(TempVisibleFlag)
        CreatePlugInWindow(TempSet, NULL);
    }

// Everything OK
//--------------
FxPtrSawData->FX_Handler_DWord_Param3 = 0;

return;
}
```

**This is one of the Main Handler routines. This is called when the host program needs the plug-in to send parameter data to the host, for example when a session file is saved. If there are no active parameter sets, the plug-in returns a buffer size of zero, telling the host that there is no data to save. The parameter set value could be a call to send just one specific parameter set or all parameter sets. The code handles both situations by first checking the parameter set value and adjusting the loop accordingly. The code first sets the 32 byte header string at the front of the buffer to be used by the plug-in to determine format information. The data for each active parameter set is stored to the buffer. Notice that control data is stored from the default data array not the current data array. If the plug-in allows automation, the current data array would have chased to the current cursor position and not reflect the starting default values. The display position and visibility data is also stored. An END_OF_LIST value is written to end the loop and the overall buffer size is returned to the host. It helps to keep the buffer size down to a minimum so the session files do not grow to huge proportions unnecessarily.**

```
//================================
// SAWStudio Handler Routine
//
// PlugIn Send Parameter Set Data
//================================

void    __stdcall   PlugInSendParamSetData(void)

{
DWORD   TempParamSet;
DWORD   TempBuffSize;
LPBYTE  TempBuffPtr;
DWORD   TempSet;
DWORD   TempMaxCount;
RECT    rWindow;

// Get Parameter Data
```

```
//-------------------
TempParamSet = FxPtrSawData->FX_Handler_DWord_Param1;
TempBuffPtr = FxPtrSawData->FX_Handler_Ptr_Param1;

// If No Active Parameter Sets... Return Zero Buffer Size
//-----------------------------------------------------
if(PlugInParamSetCount == 0)
    {
    FxPtrSawData->FX_Handler_DWord_Param2 = 0;
    return;
    }

// Set Max Count (Might Be All Sets)
//----------------------------------
if(TempParamSet == FX_ALL_PARAM_SETS)
    TempMaxCount = MAX_PARAM_SETS;
else
    TempMaxCount = 1;

lstrcpy((LPSTR)TempBuffPtr, "FX API TEST PLUGIN PRESETS 001 ");
TempBuffSize = 32;

// Save Individual Settings For Each Active ParamSet
//------------------------------------------------
for(TempSet = 0; TempSet < TempMaxCount; TempSet++)
    {
    // Skip If Not Active
    //-------------------
    if(PlugInParamSetTable[TempSet] == 0)
        continue;

    // Write Parameter Set Value
    //--------------------------
    *(PDWORD)(TempBuffPtr + TempBuffSize) = TempSet;
    TempBuffSize = TempBuffSize + 4;

    // Write PlugIn Data
    //------------------
    *(PDWORD)(TempBuffPtr + TempBuffSize) = PlugInTrack[TempSet];
    TempBuffSize = TempBuffSize + 4;

    *(PDWORD)(TempBuffPtr + TempBuffSize) = PlugInBytesPerSample[TempSet];
    TempBuffSize = TempBuffSize + 4;

    *(PDWORD)(TempBuffPtr + TempBuffSize) = PlugInDefaultDataValue[TempSet][AUTO_BYPASS_SW];
    TempBuffSize = TempBuffSize + 4;

    *(PDWORD)(TempBuffPtr + TempBuffSize) = PlugInDefaultDataValue[TempSet][AUTO_VOL_POT];
    TempBuffSize = TempBuffSize + 4;

    *(PDWORD)(TempBuffPtr + TempBuffSize) = PlugInDefaultDataValue[TempSet][AUTO_VALUE_1];
    TempBuffSize = TempBuffSize + 4;

    *(PDWORD)(TempBuffPtr + TempBuffSize) = PlugInDefaultDataValue[TempSet][AUTO_VALUE_2];
```

```
TempBuffSize = TempBuffSize + 4;

*(PDWORD)(TempBuffPtr + TempBuffSize) = PlugInDefaultDataValue[TempSet][AUTO_VALUE_3];
TempBuffSize = TempBuffSize + 4;

*(PDWORD)(TempBuffPtr + TempBuffSize) = PlugInDefaultDataValue[TempSet][AUTO_VALUE_4];
TempBuffSize = TempBuffSize + 4;

*(PDWORD)(TempBuffPtr + TempBuffSize) = PlugInDefaultDataValue[TempSet][AUTO_VALUE_5];
TempBuffSize = TempBuffSize + 4;

*(PDWORD)(TempBuffPtr + TempBuffSize) = PlugInDefaultDataValue[TempSet][AUTO_VALUE_6];
TempBuffSize = TempBuffSize + 4;

// Write Window Info
//------------------
if((hWndPlugIn[TempSet]) == 0)
    {
    // Not Visible
    //------------
    *(PDWORD)(TempBuffPtr + TempBuffSize) = 0;
    TempBuffSize = TempBuffSize + 4;

    *(PDWORD)(TempBuffPtr + TempBuffSize) = 0;
    TempBuffSize = TempBuffSize + 4;

    *(PDWORD)(TempBuffPtr + TempBuffSize) = 0;
    TempBuffSize = TempBuffSize + 4;

    *(PDWORD)(TempBuffPtr + TempBuffSize) = 0;
    TempBuffSize = TempBuffSize + 4;

    *(PDWORD)(TempBuffPtr + TempBuffSize) = 0;
    TempBuffSize = TempBuffSize + 4;
    }
else
    {
    // Visible
    //--------
    GetWindowRect(hWndPlugIn[TempSet], &rWindow);

    *(PDWORD)(TempBuffPtr + TempBuffSize) = rWindow.left;
    TempBuffSize = TempBuffSize + 4;

    *(PDWORD)(TempBuffPtr + TempBuffSize) = rWindow.top;
    TempBuffSize = TempBuffSize + 4;

    *(PDWORD)(TempBuffPtr + TempBuffSize) = rWindow.right - rWindow.left;
    TempBuffSize = TempBuffSize + 4;

    *(PDWORD)(TempBuffPtr + TempBuffSize) = rWindow.bottom - rWindow.top;
    TempBuffSize = TempBuffSize + 4;

    *(PDWORD)(TempBuffPtr + TempBuffSize) = 1;
```

```
        TempBuffSize = TempBuffSize + 4;
        }
    }

// Write End Of List
//------------------
*(PDWORD)(TempBuffPtr + TempBuffSize) = END_OF_LIST;
TempBuffSize = TempBuffSize + 4;

// Return Buffer Size
//-------------------
FxPtrSawData->FX_Handler_DWord_Param2 = TempBuffSize;

return;
}
```

**This is one of the Main Handler routines. This is called when some host operation requires the plug-in to initialize a specific parameter set's data.**

```
//================================
// SAWStudio Handler Routine
//
// PlugIn Init Parameter Set Data
//================================

void    __stdcall   PlugInInitParamSetData(void)

{
DWORD   TempParamSet;

// Get Parameter Data
//-------------------
TempParamSet = FxPtrSawData->FX_Handler_DWord_Param1;

// Init PlugIn Variables
//----------------------
InitPlugInVariables(TempParamSet, FX_ALL_CONTROLS);

return;
}
```

**This is one of the Main Handler routines. This is called whenever the session sample rate has been changed in the host. Notice that the plug-in compares the new rate to the currently stored rate, and if they are different, re-initializes all data values that might be affected.**

```
//===========================
// SAWStudio Handler Routine
//
// PlugIn Change Sample Rate
//===========================

void    __stdcall   PlugInChangeRate(void)
```

```
{
DWORD   TempParamSet;
DWORD   TempNewRate;

// Get Parameter Data
//-------------------
TempNewRate = FxPtrSawData->FX_Handler_DWord_Param1;

// Init New Data If Rate Is Different
//-----------------------------------
if(FxSampleRate != TempNewRate)
    {
    FxSampleRate = TempNewRate;

    for(TempParamSet = 0; TempParamSet < MAX_PARAM_SETS; TempParamSet++)
        {
        if(PlugInParamSetTable[TempParamSet] == 0)
            continue;

        // Init PlugIn Variables
        //----------------------
        InitPlugInVariables(TempParamSet, FX_ALL_CONTROLS);
        }
    }

return;
}
```

**This is one of the Main Handler routines. This is called whenever the session resolution has been changed in the host. Notice that the plug-in compares the new resolution to the currently stored resolution, and if they are different, re-initializes all data values that might be affected.**

```
//===========================
// SAWStudio Handler Routine
//
// PlugIn Change Resolution
//===========================

void    __stdcall   PlugInChangeRes(void)

{
DWORD   TempParamSet;
DWORD   TempNewRes;

// Get Parameter Data
//-------------------
TempNewRes = FxPtrSawData->FX_Handler_DWord_Param1;

// Init New Data If Res Is Different
//----------------------------------
if(FxResolution != TempNewRes)
    {
    FxResolution = TempNewRes;
    for(TempParamSet = 0; TempParamSet < MAX_PARAM_SETS; TempParamSet++)
        {
        if(PlugInParamSetTable[TempParamSet] == 0)
            continue;

        // Init PlugIn Variables
        //-----------------------
        InitPlugInVariables(TempParamSet, FX_ALL_CONTROLS);
        }
    }

return;
}
```

**This is one of the Main Handler routines. This is called whenever the active console channel or multitrack hot-track has been changed in the host. If the plug-in displays current track information it should be updated at this time.**

```
//============================
// SAWStudio Handler Routine
//
// PlugIn Change Active Track
//============================

void    __stdcall   PlugInChangeActiveTrack(void)

{
DWORD   TempNewTrack;

// Get Parameter Data
//--------------------
TempNewTrack = FxPtrSawData->FX_Handler_DWord_Param1;

// Perform Any Track Change Related Process Needed
//------------------------------------------------

return;
}
```

**This is one of the Main Handler routines. This is called whenever the multitrack mark begin or end position has been changed in the host. If the plug-in references this information it should be updated at this time.**

```
//===========================
// SAWStudio Handler Routine
//
// PlugIn Change MT Marks
//===========================

void    __stdcall   PlugInChangeMTMarks(void)

{
DWORD   TempNewMarkBegPos;
DWORD   TempNewMarkEndPos;

// Get Parameter Data
//-------------------
TempNewMarkBegPos = FxPtrSawData->FX_Handler_DWord_Param1;
TempNewMarkEndPos = FxPtrSawData->FX_Handler_DWord_Param2;

// Perform Any Marked Area Change Related Process Needed
//-----------------------------------------------------

return;
}
```

**This is one of the Main Handler routines. This is called whenever the Smpte Start Offset, Format or Mode has been changed in the host. If the plug-in references this information it should be updated at this time.**

```
//===========================
// SAWStudio Handler Routine
//
// PlugIn Change Smpte Info
//===========================

void    __stdcall   PlugInChangeSmpteInfo(void)

{
// The Smpte Offset Shared Memory Variables Are Set
// The Smpte Format Shared Memory Variable Is Set
// The Smpte Mode Shared Memory Variable Is Set
//
// Perform Any Smpte Info Change Related Process Needed
//----------------------------------------------------

return;
}
```

**This is one of the Main Handler routines. This is called whenever any of the Tempo variables have been changed in the host. If the plug-in references this information it should be updated at this time.**

```
//===========================
// SAWStudio Handler Routine
//
// PlugIn Change Tempo Info
//===========================

void    __stdcall   PlugInChangeTempoInfo(void)

{
// The Tempo Shared Memory Variables Are Set
//
// Perform Any Tempo Info Change Related Process Needed
//----------------------------------------------------
return;
}
```

**The next group of functions deal with the automation engine. The chase functions handle operations for chasing and updating controls every time the cursor is randomly re-positioned or layers are switched. The process functions handle updating controls during a forward moving real-time playback operation. As automation entries are encountered they are updated through these routines.**

**During a chase operation, first the plug-in data is cleared to its default values. Next, each automation entry is sent to the plug-in from the beginning position to the actual current cursor position. Finally, the plug-in is asked to display all controls which have changed from their default values. If for some reason, a track is altered and needs to be re-synced during a real-time playback operation, the same sequence is performed, except the re-sync function is sent at the end to force a variable re-initialization as well as a re-display of the final values.**

**During a normal real-time playback operation, each automation entry encountered is first processed and then later re-displayed in sync with the actual playback position.**

**This is one of the Main Handler routines. This is called whenever the multitrack cursor position is changed during idle play mode on any track that contains automation data. Any plug-in that supports automation should respond to this call to clear its current automation values and ready itself for an automation chase sequence to current position operation. This call should be skipped if there are no active parameter sets. Next it checks the parameter set value passed and adjusts the loop for one specific set or all sets. A track verification can be used as a double check on single parameter set requests to help detect left over data from another plug-in that might have been patched and removed. The current data array is saved to the automation compare data array, and the current data array is set to the default values.**

```
//===========================
// SAWStudio Handler Routine
//
// Clear Chase Automation
//===========================

void    __stdcall   PlugInClearChaseAutomation(void)

{
DWORD   TempParamSet;
DWORD   TempPhysicalTrack;
DWORD   TempSet;
DWORD   TempStartSet;
DWORD   TempEndSet;

// No Active Parameter Sets... Nothing To Do
//------------------------------------------
if(PlugInParamSetCount == 0)
    return;

// Get Parameter Data
//-------------------
TempParamSet = FxPtrSawData->FX_Handler_DWord_Param1;
TempPhysicalTrack = FxPtrSawData->FX_Handler_DWord_Param2;

// Set Start And End Loop Values (Might Be All Sets)
//--------------------------------------------------
if(TempParamSet == FX_ALL_PARAM_SETS)
```

```
    {
    TempStartSet = 0;
    TempEndSet = MAX_PARAM_SETS;
    }
else
    {
    // Verify A Track Match For This Parameter Set
    //------------------------------------------
    if(TempPhysicalTrack != PlugInTrack[TempParamSet])
        return;

    TempStartSet = TempParamSet;
    TempEndSet = TempParamSet + 1;
    }

for(TempSet = TempStartSet; TempSet < TempEndSet; TempSet++)
    {
    // Skip All Inactive Parameter Sets
    //---------------------------------
    if(PlugInParamSetTable[TempSet] == 0)
        continue;

    // Copy Current Data Values To Automation Data Storage
    //--------------------------------------------------
    CopyMemory(&PlugInAutoDataValue[TempSet][0], &PlugInDataValue[TempSet][0], MAX_AUTO_CONTROLS * 4);

    // Set Current Data Values To Default Values
    //-----------------------------------------
    CopyMemory(&PlugInDataValue[TempSet][0], &PlugInDefaultDataValue[TempSet][0], MAX_AUTO_CONTROLS * 4);
    }

return;
}
```

**This is one of the Main Handler routines. This is called whenever the multitrack cursor position is changed during idle play mode on any track that contains automation data. Any plug-in that supports automation should respond to this call to update its current data value for the specified control as the automation chase sequence to current position operation progresses. This call should be skipped if there are no active parameter sets. A track verification can be used as a double check to help detect left over data from another plug-in that might have been patched and removed. The current data array is updated to the new control value. The update should not initialize other variables or adjust the display yet, as the plug-in might receive multiple calls for the same control during the automation chase operation.**

```
//=============================
// SAWStudio Handler Routine
//
// Store Chase Automation
//=============================

void    __stdcall   PlugInStoreChaseAutomation(void)

{
DWORD   TempParamSet;
DWORD   TempPhysicalTrack;
DWORD   TempCtrlId;
int     TempDataValue;

// Get Parameter Data
//-------------------
TempParamSet = FxPtrSawData->FX_Handler_DWord_Param1;

// Skip If Inactive
//-----------------
if(PlugInParamSetTable[TempParamSet] == 0)
    return;

TempPhysicalTrack = FxPtrSawData->FX_Handler_DWord_Param2;

// Verify A Track Match For This Parameter Set
//--------------------------------------------
if(TempPhysicalTrack != PlugInTrack[TempParamSet])
    return;

// Set New Data Value
//-------------------
TempCtrlId = FxPtrSawData->FX_Handler_DWord_Param3;
TempDataValue = FxPtrSawData->FX_Handler_Int_Param1;

PlugInDataValue[TempParamSet][TempCtrlId] = TempDataValue;
return;
}
```

**This is one of the Main Handler routines. This is called whenever the multitrack cursor position is changed during idle play mode on any track that contains automation data. Any plug-in that supports automation should respond to this call to display any updated or changed controls at the end of the automation chase sequence to current position operation. This call should be skipped if there are no active parameter sets. Next it checks the parameter set value passed and adjusts the loop for one specific set or all sets. A track verification can be used as a double check on single parameter set requests to help detect left over data from another plug-in that might have been patched and removed. The current data array is compared to the previously saved automation data array and the display is updated for any control which has been changed if the parameter set window is visible.**

```
//=========================
// SAWStudio Handler Routine
//
// Display Chase Automation
//=========================

void    __stdcall   PlugInDisplayChaseAutomation(void)

{
DWORD   TempParamSet;
DWORD   TempPhysicalTrack;
DWORD   TempSet;
DWORD   TempStartSet;
DWORD   TempEndSet;
```

```
// No Active Parameter Sets... Nothing To Display
//-------------------------------------------------
if(PlugInParamSetCount == 0)
    return;

// Get Parameter Data
//-------------------
TempParamSet = FxPtrSawData->FX_Handler_DWord_Param1;
TempPhysicalTrack = FxPtrSawData->FX_Handler_DWord_Param2;

// Set Start And End Loop Values (Might Be All Sets)
//----------------------------------------------------
if(TempParamSet == FX_ALL_PARAM_SETS)
    {
    TempStartSet = 0;
    TempEndSet = MAX_PARAM_SETS;
    }
else
    {
    // Verify A Track Match For This Parameter Set
    //---------------------------------------------
    if(TempPhysicalTrack != PlugInTrack[TempParamSet])
        return;

    TempStartSet = TempParamSet;
    TempEndSet = TempParamSet + 1;
    }

for(TempSet = TempStartSet; TempSet < TempEndSet; TempSet++)
    {
    // Skip If Window Is Not Open
    //--------------------------
    if(hWndPlugIn[TempSet] == 0)
        continue;

    // Display All Changed Values
    //--------------------------
    if(PlugInDataValue[TempSet][AUTO_BYPASS_SW] != PlugInAutoDataValue[TempSet][AUTO_BYPASS_SW])
        {
        if(PlugInDataValue[TempSet][AUTO_BYPASS_SW])
            DisplayBitmapBtn(PlugInDC[TempSet], &RectPlugInBypassDisplayBtn, hPlugInBypassBtnBitmap);
        else
            RefreshBitmap(PlugInDC[TempSet], &RectPlugInBypassDisplayBtn, hPlugInMainBitmap);
        }

    if(PlugInDataValue[TempSet][AUTO_VOL_POT] != PlugInAutoDataValue[TempSet][AUTO_VOL_POT])
        DrawVertKnob(PlugInDC[TempSet], &RectPlugInVolPot, PlugInDataValue[TempSet][AUTO_VOL_POT], 1);

    if(PlugInDataValue[TempSet][AUTO_VALUE_1] != PlugInAutoDataValue[TempSet][AUTO_VALUE_1])
        DisplayPlugInValue(PlugInDC[TempSet], AUTO_VALUE_1, PlugInDataValue[TempSet][AUTO_VALUE_1], 1);

    if(PlugInDataValue[TempSet][AUTO_VALUE_2] != PlugInAutoDataValue[TempSet][AUTO_VALUE_2])
        DisplayPlugInValue(PlugInDC[TempSet], AUTO_VALUE_2, PlugInDataValue[TempSet][AUTO_VALUE_2], 1);
    if(PlugInDataValue[TempSet][AUTO_VALUE_3] != PlugInAutoDataValue[TempSet][AUTO_VALUE_3])
        DisplayPlugInValue(PlugInDC[TempSet], AUTO_VALUE_3, PlugInDataValue[TempSet][AUTO_VALUE_3], 1);

    if(PlugInDataValue[TempSet][AUTO_VALUE_4] != PlugInAutoDataValue[TempSet][AUTO_VALUE_4])
        DisplayPlugInValue(PlugInDC[TempSet], AUTO_VALUE_4, PlugInDataValue[TempSet][AUTO_VALUE_4], 1);

    if(PlugInDataValue[TempSet][AUTO_VALUE_5] != PlugInAutoDataValue[TempSet][AUTO_VALUE_5])
        DisplayPlugInValue(PlugInDC[TempSet], AUTO_VALUE_5, PlugInDataValue[TempSet][AUTO_VALUE_5], 1);

    if(PlugInDataValue[TempSet][AUTO_VALUE_6] != PlugInAutoDataValue[TempSet][AUTO_VALUE_6])
        DisplayPlugInValue(PlugInDC[TempSet], AUTO_VALUE_6, PlugInDataValue[TempSet][AUTO_VALUE_6], 1);
    }

return;
}
```

**This is one of the Main Handler routines. This is called whenever the multitrack cursor position is changed during a real-time play mode on any track that contains automation data. Any plug-in that supports automation should respond to this call to display and re-initialize any updated or changed controls at the end of the automation chase sequence to current position operation. This call should be skipped if the specified parameter set is not active. A track verification can be used as a double check on single parameter set requests to help detect left over data from another plug-in that might have been patched and removed. The current data array is compared to the previously saved automation data array. Any control which has been changed is re-initialized and the display is updated if the parameter set window is visible.**

```
//===========================
// SAWStudio Handler Routine
//
// ReSync Chase Automation
//===========================

void    __stdcall   PlugInReSyncChaseAutomation(void)

{
DWORD   TempParamSet;
DWORD   TempPhysicalTrack;

// Get Parameter Data
//-------------------
TempParamSet = FxPtrSawData->FX_Handler_DWord_Param1;

// Skip If Inactive
//-----------------
if(PlugInParamSetTable[TempParamSet] == 0)
    return;

TempPhysicalTrack = FxPtrSawData->FX_Handler_DWord_Param2;

// Verify A Track Match For This Parameter Set
//-------------------------------------------
if(TempPhysicalTrack != PlugInTrack[TempParamSet])
    return;

// Initialize And Display All Changed Values
```

```
//--------------------------------------------
if(PlugInDataValue[TempParamSet][AUTO_BYPASS_SW] != PlugInAutoDataValue[TempParamSet][AUTO_BYPASS_SW])
    {
    InitPlugInVariables(TempParamSet, AUTO_BYPASS_SW);

    // Display If Window Is Open
    //--------------------------
    if(hWndPlugIn[TempParamSet])
        {
        if(PlugInDataValue[TempParamSet][AUTO_BYPASS_SW])
            DisplayBitmapBtn(PlugInDC[TempParamSet], &RectPlugInBypassDisplayBtn, hPlugInBypassBtnBitmap);

        else
            RefreshBitmap(PlugInDC[TempParamSet], &RectPlugInBypassDisplayBtn, hPlugInMainBitmap);
        }
    }

if(PlugInDataValue[TempParamSet][AUTO_VOL_POT] != PlugInAutoDataValue[TempParamSet][AUTO_VOL_POT])
    {
    InitPlugInVariables(TempParamSet, AUTO_VOL_POT);

    // Display If Window Is Open
    //--------------------------
    if(hWndPlugIn[TempParamSet])
        DrawVertKnob(PlugInDC[TempParamSet], &RectPlugInVolPot, PlugInDataValue[TempParamSet][AUTO_VOL_POT], 1);
    }

if(PlugInDataValue[TempParamSet][AUTO_VALUE_1] != PlugInAutoDataValue[TempParamSet][AUTO_VALUE_1])
    {
    InitPlugInVariables(TempParamSet, AUTO_VALUE_1);

    // Display If Window Is Open
    //--------------------------
    if(hWndPlugIn[TempParamSet])
        DisplayPlugInValue(PlugInDC[TempParamSet], AUTO_VALUE_1, PlugInDataValue[TempParamSet][AUTO_VALUE_1], 1);
    }

if(PlugInDataValue[TempParamSet][AUTO_VALUE_2] != PlugInAutoDataValue[TempParamSet][AUTO_VALUE_2])
    {
    InitPlugInVariables(TempParamSet, AUTO_VALUE_2);

    // Display If Window Is Open
    //--------------------------
    if(hWndPlugIn[TempParamSet])
        DisplayPlugInValue(PlugInDC[TempParamSet], AUTO_VALUE_2, PlugInDataValue[TempParamSet][AUTO_VALUE_2], 1);
    }

if(PlugInDataValue[TempParamSet][AUTO_VALUE_3] != PlugInAutoDataValue[TempParamSet][AUTO_VALUE_3])
    {
    InitPlugInVariables(TempParamSet, AUTO_VALUE_3);

    // Display If Window Is Open
    //--------------------------
    if(hWndPlugIn[TempParamSet])
```

```
        DisplayPlugInValue(PlugInDC[TempParamSet], AUTO_VALUE_3, PlugInDataValue[TempParamSet][AUTO_VALUE_3], 1);
    }

if(PlugInDataValue[TempParamSet][AUTO_VALUE_4] != PlugInAutoDataValue[TempParamSet][AUTO_VALUE_4])
    {
    InitPlugInVariables(TempParamSet, AUTO_VALUE_4);

    // Display If Window Is Open
    //--------------------------
    if(hWndPlugIn[TempParamSet])
        DisplayPlugInValue(PlugInDC[TempParamSet], AUTO_VALUE_4, PlugInDataValue[TempParamSet][AUTO_VALUE_4], 1);
    }

if(PlugInDataValue[TempParamSet][AUTO_VALUE_5] != PlugInAutoDataValue[TempParamSet][AUTO_VALUE_5])
    {
    InitPlugInVariables(TempParamSet, AUTO_VALUE_5);

    // Display If Window Is Open
    //--------------------------
    if(hWndPlugIn[TempParamSet])
        DisplayPlugInValue(PlugInDC[TempParamSet], AUTO_VALUE_5, PlugInDataValue[TempParamSet][AUTO_VALUE_5], 1);
    }

if(PlugInDataValue[TempParamSet][AUTO_VALUE_6] != PlugInAutoDataValue[TempParamSet][AUTO_VALUE_6])
    {
    InitPlugInVariables(TempParamSet, AUTO_VALUE_6);

    // Display If Window Is Open
    //--------------------------
    if(hWndPlugIn[TempParamSet])
        DisplayPlugInValue(PlugInDC[TempParamSet], AUTO_VALUE_6, PlugInDataValue[TempParamSet][AUTO_VALUE_6], 1);
    }

return;
}
```

**This is one of the Main Handler routines. This is called whenever an automation entry is encountered during any active process. Any plug-in that supports automation should respond to this call to update and re-initialize its current data value for further processing at the new setting. This call should be skipped if the specified parameter set is not active. A track verification can be used as a double check to help detect left over data from another plug-in that might have been patched and removed. The current data array is updated and initialized to the new control value. The update should not be displayed at this time, because this timing occurs in front of the actual playback position.**

```
//===========================
// SAWStudio Handler Routine
//
// Change Process Automation
//===========================

void    __stdcall   PlugInChangeProcessAutomation(void)

{
DWORD   TempParamSet;
DWORD   TempPhysicalTrack;
DWORD   TempCtrlId;
int     TempDataValue;

// Get Parameter Data
//--------------------
TempParamSet = FxPtrSawData->FX_Handler_DWord_Param1;

// Skip If Inactive
//------------------
if(PlugInParamSetTable[TempParamSet] == 0)
    return;

TempPhysicalTrack = FxPtrSawData->FX_Handler_DWord_Param2;

// Verify A Track Match For This Parameter Set
//---------------------------------------------
if(TempPhysicalTrack != PlugInTrack[TempParamSet])
    return;

// Set New Data Value And Initialize It
//--------------------------------------
TempCtrlId = FxPtrSawData->FX_Handler_DWord_Param3;
TempDataValue = FxPtrSawData->FX_Handler_Int_Param1;

if(TempCtrlId == AUTO_BYPASS_SW)
    {
    PlugInDataValue[TempParamSet][AUTO_BYPASS_SW] = TempDataValue;

    InitPlugInVariables(TempParamSet, AUTO_BYPASS_SW);
    return;
    }

if(TempCtrlId == AUTO_VOL_POT)
    {
    PlugInDataValue[TempParamSet][AUTO_VOL_POT] = TempDataValue;

    InitPlugInVariables(TempParamSet, AUTO_VOL_POT);
    return;
    }

if(TempCtrlId == AUTO_VALUE_1)
    {
    PlugInDataValue[TempParamSet][AUTO_VALUE_1] = TempDataValue;

    InitPlugInVariables(TempParamSet, AUTO_VALUE_1);
    return;
    }

if(TempCtrlId == AUTO_VALUE_2)
    {
    PlugInDataValue[TempParamSet][AUTO_VALUE_2] = TempDataValue;

    InitPlugInVariables(TempParamSet, AUTO_VALUE_2);
    return;
    }

if(TempCtrlId == AUTO_VALUE_3)
    {
    PlugInDataValue[TempParamSet][AUTO_VALUE_3] = TempDataValue;

    InitPlugInVariables(TempParamSet, AUTO_VALUE_3);
    return;
    }

if(TempCtrlId == AUTO_VALUE_4)
    {
    PlugInDataValue[TempParamSet][AUTO_VALUE_4] = TempDataValue;

    InitPlugInVariables(TempParamSet, AUTO_VALUE_4);
    return;
    }

if(TempCtrlId == AUTO_VALUE_5)
    {
    PlugInDataValue[TempParamSet][AUTO_VALUE_5] = TempDataValue;

    InitPlugInVariables(TempParamSet, AUTO_VALUE_5);
    return;
    }

if(TempCtrlId == AUTO_VALUE_6)
    {
    PlugInDataValue[TempParamSet][AUTO_VALUE_6] = TempDataValue;

    InitPlugInVariables(TempParamSet, AUTO_VALUE_6);
    return;
    }

return;
}
```

**This is one of the Main Handler routines. This is called whenever an automation entry is encountered during any real-time playback operation, in sync with the actual playback position. Any plug-in that supports automation should respond to this call to update the display for the designated control. This call should be skipped if the specified parameter set is not active or the parameter set window is not visible. A track verification can be used as a double check to help detect left over data from another plug-in that might have been patched and removed. The designated control display is updated to the new data value.**

```
//============================
// SAWStudio Handler Routine
//
// Display Process Automation
//============================

void    __stdcall   PlugInDisplayProcessAutomation(void)
```

```
{
DWORD   TempParamSet;
DWORD   TempPhysicalTrack;
DWORD   TempCtrlId;
int     TempDataValue;

// Get Parameter Data
//-------------------
TempParamSet = FxPtrSawData->FX_Handler_DWord_Param1;

// Skip If Inactive
//-----------------
if(PlugInParamSetTable[TempParamSet] == 0)
    return;

// Skip If Window Is Not Open
//---------------------------
if(hWndPlugIn[TempParamSet] == 0)
    return;

TempPhysicalTrack = FxPtrSawData->FX_Handler_DWord_Param2;

// Verify A Track Match For This Parameter Set
//--------------------------------------------
if(TempPhysicalTrack != PlugInTrack[TempParamSet])
    return;

// Display Automation Changes
//---------------------------
TempCtrlId = FxPtrSawData->FX_Handler_DWord_Param3;
TempDataValue = FxPtrSawData->FX_Handler_Int_Param1;

if(TempCtrlId == AUTO_BYPASS_SW)
    {
    if(TempDataValue)
        DisplayBitmapBtn(PlugInDC[TempParamSet], &RectPlugInBypassDisplayBtn, hPlugInBypassBtnBitmap);
    else
        RefreshBitmap(PlugInDC[TempParamSet], &RectPlugInBypassDisplayBtn, hPlugInMainBitmap);

    return;
    }

if(TempCtrlId == AUTO_VOL_POT)
    {
    DrawVertKnob(PlugInDC[TempParamSet], &RectPlugInVolPot, TempDataValue, 1);
    return;
    }

if(TempCtrlId == AUTO_VALUE_1)
    {
    DisplayPlugInValue(PlugInDC[TempParamSet], AUTO_VALUE_1, TempDataValue, 1);
    return;
    }
if(TempCtrlId == AUTO_VALUE_2)
    {
    DisplayPlugInValue(PlugInDC[TempParamSet], AUTO_VALUE_2, TempDataValue, 1);
    return;
    }

if(TempCtrlId == AUTO_VALUE_3)
    {
    DisplayPlugInValue(PlugInDC[TempParamSet], AUTO_VALUE_3, TempDataValue, 1);
    return;
    }

if(TempCtrlId == AUTO_VALUE_4)
    {
    DisplayPlugInValue(PlugInDC[TempParamSet], AUTO_VALUE_4, TempDataValue, 1);
    return;
    }

if(TempCtrlId == AUTO_VALUE_5)
    {
    DisplayPlugInValue(PlugInDC[TempParamSet], AUTO_VALUE_5, TempDataValue, 1);
    return;
    }

if(TempCtrlId == AUTO_VALUE_6)
    {
    DisplayPlugInValue(PlugInDC[TempParamSet], AUTO_VALUE_6, TempDataValue, 1);
    return;
    }

return;
}
```

**This internal function initializes the plug-in. It first gets the current screen dimensions, then registers the classes it needs for window displays. Note that the class used for the plug-in display windows carries an extra 4 bytes of data with each window created. This is used by the plug-in to determine the parameter set the window is attached to. Certain array variables are initialized next. Next bitmap images are loaded and drawing tools are created. Next the Execute function message is registered for communication back to the host program. Work buffers are then allocated. The current module path is stored for later use and any special cursors it uses are loaded. Finally, threads are created for the plug-in operation. This routine sets up everything that the plug-in needs to be ready to operate.**

```
//======================
// Initialize Procedure
//======================

void    __stdcall   InitializeProc(void)

{
WNDCLASS    TempWndClass;
DWORD       TempCount;

// Get Screen Size
//-----------------
```

```
ScreenMaxH = GetSystemMetrics(SM_CXSCREEN);
ScreenMaxV = GetSystemMetrics(SM_CYSCREEN);

// Register Message Window Class
//-------------------------------
ZeroMemory((LPSTR)&TempWndClass, sizeof(WNDCLASS));

TempWndClass.style = CS_BYTEALIGNCLIENT | CS_SAVEBITS;
TempWndClass.lpfnWndProc = (WNDPROC)DLLWndMsgProc;
TempWndClass.cbClsExtra = 0;
TempWndClass.cbWndExtra = 0;
TempWndClass.hInstance = hInstFx;
TempWndClass.hIcon = NULL;
TempWndClass.hCursor = LoadCursor(NULL, IDC_ARROW);
TempWndClass.hbrBackground = (HBRUSH)GetStockObject(WHITE_BRUSH);
TempWndClass.lpszMenuName = NULL;
TempWndClass.lpszClassName = szMsgClass;

RegisterClass(&TempWndClass);

// Register PlugIn Window Class (Include DWord Of Extra Data)
//----------------------------------------------------------
ZeroMemory((LPSTR)&TempWndClass, sizeof(WNDCLASS));

TempWndClass.style = CS_BYTEALIGNCLIENT | CS_OWNDC;
TempWndClass.lpfnWndProc = (WNDPROC)DLLWndPlugInProc;
TempWndClass.cbClsExtra = 0;
TempWndClass.cbWndExtra = 4;
TempWndClass.hInstance = hInstFx;
TempWndClass.hIcon = NULL;
TempWndClass.hCursor = LoadCursor(NULL, IDC_ARROW);
TempWndClass.hbrBackground = (HBRUSH)GetStockObject(DKGRAY_BRUSH);
TempWndClass.lpszMenuName = NULL;
TempWndClass.lpszClassName = szPlugInClass;

RegisterClass(&TempWndClass);

// Init Arrays
//------------
for(TempCount = 0; TempCount < MAX_PARAM_SETS; TempCount++)
    {
    hWndPlugIn[TempCount] = 0;
    PlugInDC[TempCount] = 0;

    PlugInXStart[TempCount] = 0;
    PlugInYStart[TempCount] = 0;
    PlugInWidth[TempCount] = 0;
    PlugInHeight[TempCount] = 0;
    }

// Load BitMap Images
//-------------------
hPlugInMainBitmap = LoadBitmap(hInstFx, szPlugInMainName);
hPlugInBypassBtnBitmap = LoadBitmap(hInstFx, szPlugInBypassBtnName);
```

```
hPlugInMeterBitmap = LoadBitmap(hInstFx, szPlugInMeterName);

// Save Bitmap Header Info
//------------------------
GetObject(hPlugInMainBitmap, sizeof(BITMAP), (LPSTR)&PlugInMainBitmapHdr);
GetObject(hPlugInBypassBtnBitmap, sizeof(BITMAP), (LPSTR)&PlugInBypassBtnBitmapHdr);
GetObject(hPlugInMeterBitmap, sizeof(BITMAP), (LPSTR)&PlugInMeterBitmapHdr);

// Create Drawing Tools
//---------------------
hBlackBrush = CreateSolidBrush(RGB(0, 0, 0));
hGrayBrush = CreateSolidBrush(RGB(192, 192, 192));

hVertKnobPen = CreatePen(PS_SOLID, 1, RGB(255, 246, 137));

hBlackColor = RGB(0, 0, 0);
hTextColor = RGB(255, 246, 137);

// Register Execute Function Message
//---------------------------------
SawFxExecuteFunctionMsg = RegisterWindowMessage(szSawFxExecuteFunctionMsg);

// Allocate Work Buffers
//----------------------
PtrBuffWork1 = (LPBYTE)GlobalAlloc(GMEM_FIXED, BuffWorkSize);

if(PtrBuffWork1 == NULL)
    MemoryAllocFailureFlag = 1;

// Get Current Module Path
//------------------------
IQSGetModuleDirectory(szOrgPath);

// Load Special Cursors
//---------------------
hPlusMinusCursor = LoadCursor(hInstFx, szPlugInPlusMinusCursorName);

// Init Threads
//-------------
InitThreads();

return;
}
```

**This internal function shuts down the plug-in. It first closes all open windows. Next it shuts down the threads it created and deletes the bitmap objects. Next it frees the work buffers and deletes the drawing tool objects. Finally it deletes any special cursor objects. At this point the plug-in is ready to be released from memory by the host program.**

```
//====================
// Shutdown Procedure
//====================
void    __stdcall   ShutdownProc(void)

{
DWORD   TempCount;

// Close All Open Windows
//------------------------
for(TempCount = 0; TempCount < MAX_PARAM_SETS; TempCount++)
    {
    if(hWndPlugIn[TempCount])
        DestroyPlugInWindow(TempCount);
    }

// ShutDown Threads
//------------------
ShutDownThreads();

// Delete Bitmaps
//----------------
DeleteObject(hPlugInMainBitmap);
DeleteObject(hPlugInBypassBtnBitmap);
DeleteObject(hPlugInMeterBitmap);

// Free Buffers
//--------------
if(PtrBuffWork1)
    GlobalFree(PtrBuffWork1);

// Delete Drawing Tools
//----------------------
DeleteObject(hBlackBrush);
DeleteObject(hGrayBrush);

DeleteObject(hVertKnobPen);

// Destroy Special Cursors
//-------------------------
DestroyCursor(hPlusMinusCursor);

return;
}
```

**This next group of functions handles internal operations used to support the plug-in, but not directly tied to the API explanation. The details of these functions can be found in the actual source .cpp file.**

```
void    __stdcall   InitThreads(void)
void    __stdcall   ShutDownThreads(void)
void    __stdcall   ForceThreadExit(HANDLE TempThreadHandle, DWORD TempThreadID, PDWORD TempExitFlagPtr)
DWORD   WINAPI      FxBackServiceCallback(LPVOID lParam)

void    __stdcall   IQSGetModuleDirectory(LPSTR TempPath)
void    __stdcall   ExtractPath(LPSTR PtrPathBuffer)
void    __stdcall   IQSGetCurDirectory(LPSTR TempPath)
void    __stdcall   AddBackSlashToPath(LPSTR PtrPathBuffer)

void    __stdcall   CreateAboutWindow(void)
void    __stdcall   PaintAboutWindow(HDC hDC)

LRESULT CALLBACK    DLLWndMsgProc(HWND hWnd, UINT wMessage, WPARAM wParam, LPARAM lParam)

int     __stdcall   CheckBoundaries(LPRECT RectAddr)

BOOL    __stdcall   PickOpenFileName(LPSTR szTempStartName, LPSTR szTempStartPath, LPSTR szTempFilterList,
                    int TempFilterIndex, LPSTR szTempDefExt)

BOOL    __stdcall   PickSaveFileName(LPSTR szTempStartName, LPSTR szTempStartPath, LPSTR szTempFilterList,
                    int TempFilterIndex, LPSTR szTempDefExt, DWORD TempOverWriteFlag)

void    __stdcall   RefreshBitmap(HDC hDC, LPRECT RectAddr, HBITMAP hBitmap)
void    __stdcall   DisplayBitmapBtn(HDC hDC, LPRECT RectAddr, HBITMAP hBitmapBtn)
DWORD   __stdcall   ChkVKey(DWORD VKeyType)
void    __stdcall   IQSMoveWindow(HWND hWnd)
void    __stdcall   GradientFillRect(HDC hDC, LPRECT RectAddr, int RectBottom, int ColorR, int ColorG, int ColorB)
void    __stdcall   ClearAllMouseMsgs(void)
int     __stdcall   WaitMsecForEvent(int DelayTime, UINT Event)
```

**This function creates and displays an individual parameter set window. This window is the interface that the user interacts with to adjust the various settings and controls of the plug-in. If the window is already open, the code simply refreshes the display. If the PlugInWidth variable is zero, the plug-in size and position is calculated. If a reference window handle has been passed, it uses this as a position template, otherwise it centers the window on the screen. If the size and position variables already contain values, then the window is created using those values. The parameter set value is saved into the extra DWORD location with each window. The display DC is initialized and the window is displayed.**

```
//======================
// Create PlugIn Window
//======================

void    __stdcall   CreatePlugInWindow(DWORD TempParamSet, HWND hWndRefWindow)

{
RECT    TempRect;

// If Window Is Already Opened Then Just Refresh It
//-------------------------------------------------
if(hWndPlugIn[TempParamSet])
    {
    InvalidateRect(hWndPlugIn[TempParamSet], NULL, FALSE);
    UpdateWindow(hWndPlugIn[TempParamSet]);
    return;
    }

// If This Is The First Time Then Use The Reference Window For Positioning
// Or Center It If There Is No Reference Window Handle
//------------------------------------------------------------------------
if(PlugInWidth[TempParamSet] == 0)
    {
```

```
    PlugInWidth[TempParamSet] = PlugInMainBitmapHdr.bmWidth;
    PlugInHeight[TempParamSet] = PlugInMainBitmapHdr.bmHeight;

    if(hWndRefWindow)
        {
        GetWindowRect(hWndRefWindow, &TempRect);
        PlugInXStart[TempParamSet] = TempRect.left + 40;
        PlugInYStart[TempParamSet] = TempRect.top + 40;
        }
    else
        {
        PlugInXStart[TempParamSet] = (ScreenMaxH - PlugInWidth[TempParamSet]) / 2;
        PlugInYStart[TempParamSet] = 80;
        }

    if(PlugInXStart[TempParamSet] + PlugInWidth[TempParamSet] > ScreenMaxH)
        PlugInXStart[TempParamSet] = ScreenMaxH - PlugInWidth[TempParamSet] - 4;

    if(PlugInYStart[TempParamSet] + PlugInHeight[TempParamSet] > ScreenMaxV)
        PlugInYStart[TempParamSet] = ScreenMaxV - PlugInHeight[TempParamSet] - 4;
    }

// Create Window
//--------------
hWndPlugIn[TempParamSet] = CreateWindowEx(
                WS_EX_TOPMOST,
                szPlugInClass,
                "",

                WS_POPUP        |
                WS_CLIPCHILDREN,

                PlugInXStart[TempParamSet],
                PlugInYStart[TempParamSet],
                PlugInWidth[TempParamSet],
                PlugInHeight[TempParamSet],

                hWndSawMain,
                NULL,
                hInstFx,
                NULL);

if(hWndPlugIn[TempParamSet] == NULL)
    return;

// Set ParamSet Value As Extra Data
//---------------------------------
SetWindowLong(hWndPlugIn[TempParamSet], 0, TempParamSet);

// Set DC Info
//------------
PlugInDC[TempParamSet] = GetDC(hWndPlugIn[TempParamSet]);
SetBkMode(PlugInDC[TempParamSet], TRANSPARENT);
SelectObject(PlugInDC[TempParamSet], hIQSSysFont);
```

```
// Display The Window And Give It The Focus
//-----------------------------------------
ShowWindow(hWndPlugIn[TempParamSet], SW_SHOW);
UpdateWindow(hWndPlugIn[TempParamSet]);
SetFocus(hWndPlugIn[TempParamSet]);
return;
}
```

**This function closes and destroys the parameter set window. The size and position are saved and used the next time the same parameter set window is opened.**

```
//=======================
// Destroy PlugIn Window
//=======================

void    __stdcall   DestroyPlugInWindow(DWORD TempParamSet)

{
RECT    TempRect;

// Skip If Window Is Already Destroyed
//------------------------------------
if(hWndPlugIn[TempParamSet] == 0)
    return;

// Save The Last Position Info
//----------------------------
GetWindowRect(hWndPlugIn[TempParamSet], &TempRect);
PlugInXStart[TempParamSet] = TempRect.left;
PlugInYStart[TempParamSet] = TempRect.top;
PlugInWidth[TempParamSet] = TempRect.right - TempRect.left;
PlugInHeight[TempParamSet] = TempRect.bottom - TempRect.top;

// Destroy Window
//---------------
ReleaseDC(hWndPlugIn[TempParamSet], PlugInDC[TempParamSet]);
DestroyWindow(hWndPlugIn[TempParamSet]);

hWndPlugIn[TempParamSet] = 0;
PlugInDC[TempParamSet] = 0;
return;
}
```

**This function processes the window messages. The current active parameter set value is set each time a message comes through by reading the extra DWORD location saved when the window was first created. The paint message displays the plug-in bitmap, displays the track assignment info, presets any buttons that are pushed in, and displays the knobs, parameter values and labels. This routine also routes each mouse click and keyboard command to the proper function.**

```
//=================================================
// PlugIn Window Procedure For Processing Messages
```

```
//==================================================

LRESULT CALLBACK    DLLWndPlugInProc(HWND hWnd, UINT wMessage, WPARAM wParam, LPARAM lParam)

{
HDC         hDC;
PAINTSTRUCT psPaint;
HDC         TempMemoryDC;

// Query Window For Parameter Set Value
//------------------------------------
FxActiveParamSet = GetWindowLong(hWnd, 0);

switch(wMessage)
    {
    case WM_PAINT:
        hDC = BeginPaint(hWnd, &psPaint);

        // Put Bitmap Onscreen
        //---------------------
        TempMemoryDC = CreateCompatibleDC(hDC);
        SelectObject(TempMemoryDC, hPlugInMainBitmap);
        BitBlt(hDC, 0, 0, PlugInMainBitmapHdr.bmWidth, PlugInMainBitmapHdr.bmHeight, TempMemoryDC, 0, 0, SRCCOPY);
        DeleteDC(TempMemoryDC);

        // Display Track Number
        //----------------------
        SetTextColor(hDC, hTextColor);

        if(PlugInTrack[FxActiveParamSet] == NO_ASSIGNMENT)
            lstrcpy(szWorkBuff, "- - -");
        else
            {
            if(PlugInTrack[FxActiveParamSet] < FxReturnTrkOffset)
                {
                wsprintf(szWorkBuff, "In %02u", PlugInTrack[FxActiveParamSet] + 1);
                goto DisplayTrack;
                }

            if(PlugInTrack[FxActiveParamSet] < FxOutputTrkOffset)
                {
                wsprintf(szWorkBuff, "Rtn %02u", PlugInTrack[FxActiveParamSet] - FxReturnTrkOffset + 1);
                goto DisplayTrack;
                }

            wsprintf(szWorkBuff, "Out %02u", PlugInTrack[FxActiveParamSet] - FxOutputTrkOffset + 1);
            }

        DisplayTrack:
        DrawText(hDC, szWorkBuff, -1, &RectPlugInTrack, DT_SINGLELINE | DT_CENTER | DT_VCENTER | DT_WORDBREAK);

        // Preset Buttons
        //---------------
        PresetPlugInBtns(0);
        // Draw Knobs
        //-----------
        DrawVertKnob(hDC, &RectPlugInVolPot, PlugInDataValue[FxActiveParamSet][AUTO_VOL_POT], 0);

        // Display Values
        //---------------
        DisplayPlugInValue(hDC, AUTO_VALUE_1, PlugInDataValue[FxActiveParamSet][AUTO_VALUE_1], 0);
        DisplayPlugInValue(hDC, AUTO_VALUE_2, PlugInDataValue[FxActiveParamSet][AUTO_VALUE_2], 0);
        DisplayPlugInValue(hDC, AUTO_VALUE_3, PlugInDataValue[FxActiveParamSet][AUTO_VALUE_3], 0);
        DisplayPlugInValue(hDC, AUTO_VALUE_4, PlugInDataValue[FxActiveParamSet][AUTO_VALUE_4], 0);
        DisplayPlugInValue(hDC, AUTO_VALUE_5, PlugInDataValue[FxActiveParamSet][AUTO_VALUE_5], 0);
        DisplayPlugInValue(hDC, AUTO_VALUE_6, PlugInDataValue[FxActiveParamSet][AUTO_VALUE_6], 0);

        // Display Labels
        //---------------
        SetTextColor(hDC, hBlackColor);
        DrawText(hDC, PlugInParamLabel[0], -1, &RectPlugInLabel1, DT_SINGLELINE | DT_CENTER | DT_VCENTER);
        DrawText(hDC, PlugInParamLabel[1], -1, &RectPlugInLabel2, DT_SINGLELINE | DT_CENTER | DT_VCENTER);
        DrawText(hDC, PlugInParamLabel[2], -1, &RectPlugInLabel3, DT_SINGLELINE | DT_CENTER | DT_VCENTER);
        DrawText(hDC, PlugInParamLabel[3], -1, &RectPlugInLabel4, DT_SINGLELINE | DT_CENTER | DT_VCENTER);
        DrawText(hDC, PlugInParamLabel[4], -1, &RectPlugInLabel5, DT_SINGLELINE | DT_CENTER | DT_VCENTER);
        DrawText(hDC, PlugInParamLabel[5], -1, &RectPlugInLabel6, DT_SINGLELINE | DT_CENTER | DT_VCENTER);

        EndPaint(hWnd, &psPaint);
        break;

    case WM_SETCURSOR:
        // Maintain Plus-Minus Cursor If Needed
        //-------------------------------------
        if(PlusMinusActive)
            break;

        goto DefProc;
        break;

    case WM_CLOSE:
        DestroyPlugInWindow(FxActiveParamSet);
        break;

    case WM_LBUTTONDOWN:
        CurMouseX = LOWORD(lParam);
        CurMouseY = HIWORD(lParam);

        // Check If Mouse Click Is In Any Active Zones
        //--------------------------------------------
        CheckPlugInZones();
        break;

    case WM_LBUTTONUP:
        ClipCursor(NULL);

        // End Plus-Minus If Needed
        //-------------------------
        if(PlusMinusActive)
            PlusMinusEnd();

        // End Vertical Knob Motion If Needed
        //-----------------------------------
        if(VertKnobActive)
            {
            // Set New Value
            //--------------
            SetVertKnobValue();

            VertKnobActive = 0;
            VertKnobCtrlId = NO_VALUE;
            SendMessage(hWndSawMain, SawFxExecuteFunctionMsg, SET_EDL_MODIFY_FLAG, 0);
            }

        break;

    case WM_RBUTTONDOWN:
        CurMouseX = LOWORD(lParam);
        CurMouseY = HIWORD(lParam);

        // Set To Default Value If Needed
        //-------------------------------
        if(PlusMinusActive)
            {
            PlusMinusActive = 0;
            FxBackThreadSuspendFlag = 1;
            ActionSetToDefaultValue();
            break;
            }

        if(VertKnobActive)
            {
            SetVertKnobValueDef();
            break;
            }

        // Check For Right-Click Active Zones
        //-----------------------------------
        if(CheckPlugInZonesRight())
            break;

        // Start Or Stop Playback As Needed
        //---------------------------------
        if(FxProcessFlag == REALTIME_PLAYBACK)
            SendMessage(hWndSawMain, SawFxExecuteFunctionMsg, STOP_MT_AUDIO, 0);
        else
            SendMessage(hWndSawMain, SawFxExecuteFunctionMsg, START_MT_AUDIO_PLAY, 0);

        break;

    case WM_MOUSEMOVE:
        CurMouseX = LOWORD(lParam);
        CurMouseY = HIWORD(lParam);

        // Adjust Knob Value
        //-------------------
        if(VertKnobActive)
            {
            MoveVertKnob();

            if(FxProcessFlag == REALTIME_PLAYBACK)
                SetVertKnobValue();
            }

        break;

    DefProc:
    default:
        return DefWindowProc(hWnd, wMessage, wParam, lParam);
    }

return (0L);
}
```

**This next group of functions handles internal operations used to support the plug-in, but not directly tied to the API explanation. The details of these functions can be found in the actual source .cpp file.**

```
void    __stdcall   DisplayPlugInValue(HDC hDC, DWORD TempCtrlId, int TempDataValue, DWORD RefreshFlag)
void    __stdcall   PresetPlugInBtns(DWORD RefreshFlag)
void    __stdcall   CheckPlugInZones(void)
DWORD   __stdcall   CheckPlugInZonesRight(void)

void    __stdcall   DrawVertKnob(HDC hDC, LPRECT RectAddr, DWORD CenterPos, DWORD RefreshFlag)
void    __stdcall   LockToVertKnob(void)
void    __stdcall   MoveVertKnob(void)
void    __stdcall   SetVertKnobValue(void)
void    __stdcall   SetVertKnobValueDef(void)

void    __stdcall   PlusMinusCheck(void)
void    __stdcall   PlusMinusEnd(void)
void    __stdcall   ActionControlScroll(void)
void    __stdcall   MoveControlScroll(void)
void    __stdcall   ActionControlMenuList(void)
void    __stdcall   ActionSetToDefaultValue(void)
void    __stdcall   ActionUpdateControl(void)

void    __stdcall   CreateOptionsSelectionMenu(void)
```

**This function sends a control change to the automation engine if the automation write mode is active. Otherwise, the new control value is set as the new default value.**

```
//=================
// Send Automation
```

```
//=================

void    __stdcall   SendAutomation(DWORD TempCtrlId, int TempDataValue, int TempMinDataValue, int TempMaxDataValue)

{
// Set As New Default Value If Automation Mode Is Not Active
//-------------------------------------------------------------
if(FxPtrSawData->FX_MTAutomationWriteFlag == 0)
    {
    PlugInDefaultDataValue[FxActiveParamSet][TempCtrlId] = TempDataValue;
    return;
    }

// Send To Automation
//--------------------
FxPtrSawData->FX_Function_DWord_Param1 = FxPlugInIndex;
FxPtrSawData->FX_Function_DWord_Param2 = FxActiveParamSet;
FxPtrSawData->FX_Function_DWord_Param3 = PlugInTrack[FxActiveParamSet];
FxPtrSawData->FX_Function_DWord_Param4 = TempCtrlId;
FxPtrSawData->FX_Function_Int_Param1 = TempDataValue;
FxPtrSawData->FX_Function_Int_Param2 = TempMinDataValue;
FxPtrSawData->FX_Function_Int_Param3 = TempMaxDataValue;

SendMessage(hWndSawMain, SawFxExecuteFunctionMsg, SEND_TO_AUTOMATION, 0);
return;
}
```

**This function saves a single parameter set as a preset file.**

```
//=====================
// Save PlugIn Preset
//=====================

void    __stdcall   SavePlugInPreset(void)

{
BOOL    Result;
int     TempFileHandle;
LPBYTE  TempDataPtr;
DWORD   TempDataSet;
DWORD   TempDataBuffSize;

// Select FileName
//-----------------
lstrcpy((LPSTR)szStringBuff, "Save PlugIn Preset");
Result = PickSaveFileName(LastPlugInFileName, szDefaultPath, szAllFilters, PLG_FILE, szPlgQual, 1);

// Canceled
//----------
if(Result == 0)
    return;

// Save New Settings
//------------------
IQSGetCurDirectory(szDefaultPath);
lstrcpy(LastPlugInFileName, szSaveName + ofFileName.nFileOffset);

lstrcpy(szWork1Name, szSaveName);

// Create New File Or OverWrite Old One
//-------------------------------------
TempFileHandle = OpenFile(szWork1Name, &ofFileStructure, OF_CREATE);

if(TempFileHandle == HFILE_ERROR)
    return;

// Organize Data In Memory Buffer
//-------------------------------
TempDataPtr = PtrBuffWork1;
TempDataSet = FxActiveParamSet;
TempDataBuffSize = 0;

// Save Header
//------------
lstrcpy((LPSTR)TempDataPtr, "FX API TEST PLUG-IN PRESET 001  ");
TempDataBuffSize = TempDataBuffSize + 32;

// Save PlugIn Current Data
//-------------------------
*(PDWORD)(TempDataPtr + TempDataBuffSize) = PlugInDataValue[TempDataSet][AUTO_VOL_POT];
TempDataBuffSize = TempDataBuffSize + 4;

*(PDWORD)(TempDataPtr + TempDataBuffSize) = PlugInDataValue[TempDataSet][AUTO_VALUE_1];
TempDataBuffSize = TempDataBuffSize + 4;

*(PDWORD)(TempDataPtr + TempDataBuffSize) = PlugInDataValue[TempDataSet][AUTO_VALUE_2];
TempDataBuffSize = TempDataBuffSize + 4;

*(PDWORD)(TempDataPtr + TempDataBuffSize) = PlugInDataValue[TempDataSet][AUTO_VALUE_3];
TempDataBuffSize = TempDataBuffSize + 4;

*(PDWORD)(TempDataPtr + TempDataBuffSize) = PlugInDataValue[TempDataSet][AUTO_VALUE_4];
TempDataBuffSize = TempDataBuffSize + 4;

*(PDWORD)(TempDataPtr + TempDataBuffSize) = PlugInDataValue[TempDataSet][AUTO_VALUE_5];
TempDataBuffSize = TempDataBuffSize + 4;

*(PDWORD)(TempDataPtr + TempDataBuffSize) = PlugInDataValue[TempDataSet][AUTO_VALUE_6];
TempDataBuffSize = TempDataBuffSize + 4;

// Write Data To Disk
//-------------------
_lwrite(TempFileHandle, (LPSTR)TempDataPtr, TempDataBuffSize);
_lclose(TempFileHandle);
return;
}
```

**This function loads a single parameter set from a previously saved preset file. The new data is loaded to the default data array and then copied to the current data array. The data is initialized and the display is updated if the window is open.**

```
//====================
// Load PlugIn Preset
//====================

void    __stdcall   LoadPlugInPreset(void)

{
DWORD   TempByteCount;
int     TempFileHandle;
BOOL    Result;
LPBYTE  TempDataPtr;
DWORD   TempDataSet;

// Select FileName
//----------------
lstrcpy((LPSTR)szStringBuff, "Load PlugIn Preset");
Result = PickOpenFileName(LastPlugInFileName, szDefaultPath, szAllFilters, PLG_FILE, szPlgQual);

// Canceled
//---------
if(Result == 0)
    return;

// Save New Settings
//------------------
IQSGetCurDirectory(szDefaultPath);
lstrcpy(LastPlugInFileName, szOpenName + ofFileName.nFileOffset);

lstrcpy(szWork1Name, szOpenName);

// Read File Into Memory
//----------------------
TempFileHandle = OpenFile(szWork1Name, &ofFileStructure, OF_READ);

if(TempFileHandle == HFILE_ERROR)
    return;

TempDataPtr = PtrBuffWork1;
TempDataSet = FxActiveParamSet;

// Ask For More Data Than The File Size Will Ever Be
//--------------------------------------------------
_lread(TempFileHandle, TempDataPtr, 0x0000f000);
_lclose(TempFileHandle);

// Check For Proper File Format
//----------------------------
if(lstrcmp((LPSTR)TempDataPtr, "FX API TEST PLUG-IN PRESET 001  ") != 0)
    {
    MessageBox(NULL, ImproperPresetMsg, WarningMsg, MB_SYSTEMMODAL | MB_ICONEXCLAMATION | MB_OK);
    return;
    }

// Set Default Variables From Memory Data
//---------------------------------------
TempByteCount = 32;

PlugInDefaultDataValue[TempDataSet][AUTO_VOL_POT] = *(PDWORD)(TempDataPtr + TempByteCount);
TempByteCount = TempByteCount + 4;

PlugInDefaultDataValue[TempDataSet][AUTO_VALUE_1] = *(PDWORD)(TempDataPtr + TempByteCount);
TempByteCount = TempByteCount + 4;

PlugInDefaultDataValue[TempDataSet][AUTO_VALUE_2] = *(PDWORD)(TempDataPtr + TempByteCount);
TempByteCount = TempByteCount + 4;

PlugInDefaultDataValue[TempDataSet][AUTO_VALUE_3] = *(PDWORD)(TempDataPtr + TempByteCount);
TempByteCount = TempByteCount + 4;

PlugInDefaultDataValue[TempDataSet][AUTO_VALUE_4] = *(PDWORD)(TempDataPtr + TempByteCount);
TempByteCount = TempByteCount + 4;

PlugInDefaultDataValue[TempDataSet][AUTO_VALUE_5] = *(PDWORD)(TempDataPtr + TempByteCount);
TempByteCount = TempByteCount + 4;

PlugInDefaultDataValue[TempDataSet][AUTO_VALUE_6] = *(PDWORD)(TempDataPtr + TempByteCount);
TempByteCount = TempByteCount + 4;

// Set All Current Values To The New Defaults
//-------------------------------------------
CopyMemory(&PlugInDataValue[TempDataSet][0], &PlugInDefaultDataValue[TempDataSet][0], MAX_AUTO_CONTROLS * 4);

// Init PlugIn Variables
//----------------------
InitPlugInVariables(TempDataSet, FX_ALL_CONTROLS);

// Update The Display If Needed
//-----------------------------
if(hWndPlugIn[TempDataSet])
    {
    InvalidateRect(hWndPlugIn[TempDataSet], NULL, FALSE);
    UpdateWindow(hWndPlugIn[TempDataSet]);
    }

return;
}
```

**This function initializes individual control data or all control data in a parameter set. Any calculated variables based on the individual control data values is initialized here. The control ID might be FX_ALL_CONTROLS or an individual control type.**

```
//======================
// Init PlugIn Variables
//======================
void    __stdcall   InitPlugInVariables(DWORD TempParamSet, DWORD TempControlID)

{
DWORD   TempIndex;

// Init All Controls For This Parameter Set
//----------------------------------------
if(TempControlID == FX_ALL_CONTROLS)
    {
    // Clear Meters If Plug-In Is Bypassed During RealTime Play
    //-------------------------------------------------------
    if(FxProcessFlag == REALTIME_PLAYBACK)
        {
        if(PlugInDataValue[TempParamSet][AUTO_BYPASS_SW])
            ResetMeterDisplay(TempParamSet);
        }

    // Calculate Volume Factor
    //------------------------
    FxVolFactor[TempParamSet] = 105 - PlugInDataValue[TempParamSet][AUTO_VOL_POT];

    // Set Output Format
    //------------------
    FxOutputFormat[TempParamSet] = PlugInDataValue[TempParamSet][AUTO_VALUE_1];

    // Set Meter Decay Value
    //----------------------
    TempIndex = PlugInDataValue[TempParamSet][AUTO_VALUE_2];
    PlugInMeterDecayFactor[TempParamSet] = MeterResponseValue[TempIndex];
    return;
    }

// Init Specified Control Only For This Parameter Set
//---------------------------------------------------
if(TempControlID == AUTO_BYPASS_SW)
    {
    // Clear Meters If Plug-In Is Bypassed During RealTime Play
    //-------------------------------------------------------
    if(FxProcessFlag == REALTIME_PLAYBACK)
        {
        if(PlugInDataValue[TempParamSet][AUTO_BYPASS_SW])
            ResetMeterDisplay(TempParamSet);
        }
    }

if(TempControlID == AUTO_VOL_POT)
    {
    // Calculate Volume Factor
    //------------------------
    FxVolFactor[TempParamSet] = 105 - PlugInDataValue[TempParamSet][AUTO_VOL_POT];
    return;
    }
if(TempControlID == AUTO_VALUE_1)
    {
    // Set Output Format
    //------------------
    FxOutputFormat[TempParamSet] = PlugInDataValue[TempParamSet][AUTO_VALUE_1];
    return;
    }

if(TempControlID == AUTO_VALUE_2)
    {
    // Set Meter Decay Value
    //----------------------
    TempIndex = PlugInDataValue[TempParamSet][AUTO_VALUE_2];
    PlugInMeterDecayFactor[TempParamSet] = MeterResponseValue[TempIndex];
    return;
    }

return;
}
```

**This function initializes the meter variables for a specific parameter set.**

```
//===================
// Init PlugIn Meter
//===================

void    __stdcall   InitPlugInMeter(DWORD TempParamSet)

{
// Init Meter Variables
//---------------------
PlugInMeterIndexHead[TempParamSet] = 0;
PlugInMeterIndexTail[TempParamSet] = 0;

PlugInMeterLastValueL[TempParamSet] = 0;
PlugInMeterLastValueR[TempParamSet] = 0;

PlugInMeterPos[TempParamSet][0] = END_OF_LIST;
PlugInMeterValueL[TempParamSet][0] = 0;
PlugInMeterValueR[TempParamSet][0] = 0;
return;
}
```

**This function resets an individual or all parameter sets to default values. It first checks the parameter set value passed and adjusts the loop for one specific set or all sets. The parameter set count and patch table are updated accordingly. Both the default data array and the current data array are set to the plug-in default starting values.**

```
//============================
// Reset PlugIn Parameter Set
//============================
void    __stdcall   ResetParamSet(DWORD TempParamSet)

{
DWORD   TempSet;
DWORD   TempStartSet;
DWORD   TempEndSet;

// Set Start And End Indexes (Might Be All Sets)
//----------------------------------------------
if(TempParamSet == FX_ALL_PARAM_SETS)
    {
    TempStartSet = 0;
    TempEndSet = MAX_PARAM_SETS;

    PlugInParamSetCount = 0;
    }
else
    {
    TempStartSet = TempParamSet;
    TempEndSet = TempStartSet + 1;

    if(PlugInParamSetCount)
        PlugInParamSetCount = PlugInParamSetCount - 1;
    }

// Loop And Reset All Variables For Each Parameter Set
//----------------------------------------------------
for(TempSet = TempStartSet; TempSet < TempEndSet; TempSet++)
    {
    PlugInParamSetTable[TempSet] = 0;
    PlugInTrack[TempSet] = NO_ASSIGNMENT;
    PlugInBytesPerSample[TempSet] = 4;

    PlugInDefaultDataValue[TempSet][AUTO_BYPASS_SW] = 0;
    PlugInDataValue[TempSet][AUTO_BYPASS_SW] = 0;

    PlugInDefaultDataValue[TempSet][AUTO_VOL_POT] = 5;
    PlugInDataValue[TempSet][AUTO_VOL_POT] = 5;

    PlugInDefaultDataValue[TempSet][AUTO_VALUE_1] = 0;
    PlugInDataValue[TempSet][AUTO_VALUE_1] = 0;

    PlugInDefaultDataValue[TempSet][AUTO_VALUE_2] = 0;
    PlugInDataValue[TempSet][AUTO_VALUE_2] = 0;

    PlugInDefaultDataValue[TempSet][AUTO_VALUE_3] = 0;
    PlugInDataValue[TempSet][AUTO_VALUE_3] = 0;

    PlugInDefaultDataValue[TempSet][AUTO_VALUE_4] = 0;
    PlugInDataValue[TempSet][AUTO_VALUE_4] = 0;

    PlugInDefaultDataValue[TempSet][AUTO_VALUE_5] = 0;
    PlugInDataValue[TempSet][AUTO_VALUE_5] = 0;
    PlugInDefaultDataValue[TempSet][AUTO_VALUE_6] = 0;
    PlugInDataValue[TempSet][AUTO_VALUE_6] = 0;
    }

return;
}
```

**This function does the actual buffer processing on the 32 bit data streams passed to the plug-in by the host program. It first checks the buffer size and returns if there is nothing to process. This could happen because a previous plug-in on this track withheld the last buffer. The routine next processes the volume of the data in place. The data format is 4 bytes per sample per channel which equates to 8 bytes total per sample. Each of the left and right samples is adjusted by the volume factor and replaced in the buffer. Next it compares the plug-in output format and performs the proper adjustment, again keeping the results in place. The last portion of the routine scans the buffer and collects the highest peak value for each channel to be used for the meter display. The values are scaled to an actual display segment value based on the meter display bitmap, and compared to the last peak values. If the new values are below the last values, then analog decay is simulated by subtracting a small decay factor from each value. This controls the actual response time of the visual meter display. The values are stored in the meter variable arrays along with the actual process position of this buffer at the head count position. Since this function is called in front of the actual playback position, we do not display the results at this time. The results are displayed in sync with the actual playback position in the FxChangePosition function call.**

**For highest performance, it is recommended that this routine be written in assembly language, if you can. If your processing algorithms require floating point buffer data, you can add a small code loop at the start of this routine to copy each sample into a float work buffer, process there in floating point and then copy the results back as integers into this buffer.**

```
//==================================
// Do PlugIn Process 32 Bit Stereo
//
// 4 Bytes Per Sample Per Channel
//==================================

void    __stdcall   DoPlugInProcess32(void)

{
int     TempBuffSize;
int     TempValue;
int     TempValueL;
int     TempValueR;
int     TempPeakValueL;
int     TempPeakValueR;
DWORD   TempIndex;
LPBYTE  TempDataPtr;

// Skip If There Is Nothing To Process
//-----------------------------------
if(FxProcessBuffSize == 0)
    return;

// Do Volume Process
//------------------
TempBuffSize = FxProcessBuffSize;
TempDataPtr = FxProcessBuffPtr;
```

```
while(TRUE)
    {
    // Adjust Value Of Each Sample
    //---------------------------
    TempValue = *(PINT)(TempDataPtr);
    TempValue = (TempValue * FxVolFactor[FxProcessParamSet]) / 100;
    *(PINT)(TempDataPtr) = TempValue;

    TempValue = *(PINT)(TempDataPtr + 4);
    TempValue = (TempValue * FxVolFactor[FxProcessParamSet]) / 100;
    *(PINT)(TempDataPtr + 4) = TempValue;

    TempDataPtr = TempDataPtr + 8;
    TempBuffSize = TempBuffSize - 8;

    if(TempBuffSize <= 0)
        break;
    }

// Do Mono Format
//---------------
if(FxOutputFormat[FxProcessParamSet] == 1)
    {
    TempBuffSize = FxProcessBuffSize;
    TempDataPtr = FxProcessBuffPtr;

    while(TRUE)
        {
        // Adjust Left Value Down 6db
        //---------------------------
        TempValueL = *(PINT)(TempDataPtr);
        TempValueL = TempValueL / 2;

        // Adjust Right Value Down 6db
        //----------------------------
        TempValueR = *(PINT)(TempDataPtr + 4);
        TempValueR = TempValueR / 2;

        // Add Them Together And Store The Result
        //---------------------------------------
        TempValue = TempValueL + TempValueR;
        *(PINT)(TempDataPtr) = TempValue;
        *(PINT)(TempDataPtr + 4) = TempValue;

        TempDataPtr = TempDataPtr + 8;
        TempBuffSize = TempBuffSize - 8;

        if(TempBuffSize <= 0)
            break;
        }
    }

// Do Lft Only Format
//-------------------
```

```
if(FxOutputFormat[FxProcessParamSet] == 2)
    {
    TempBuffSize = FxProcessBuffSize;
    TempDataPtr = FxProcessBuffPtr;

    while(TRUE)
        {
        // Clear All Right Channel Samples
        //-------------------------------
        *(PINT)(TempDataPtr + 4) = 0;

        TempDataPtr = TempDataPtr + 8;
        TempBuffSize = TempBuffSize - 8;

        if(TempBuffSize <= 0)
            break;
        }
    }

// Do Rgt Only Format
//-------------------
if(FxOutputFormat[FxProcessParamSet] == 3)
    {
    TempBuffSize = FxProcessBuffSize;
    TempDataPtr = FxProcessBuffPtr;

    while(TRUE)
        {
        // Clear All Left Channel Samples
        //------------------------------
        *(PINT)(TempDataPtr) = 0;

        TempDataPtr = TempDataPtr + 8;
        TempBuffSize = TempBuffSize - 8;

        if(TempBuffSize <= 0)
            break;
        }
    }

// Do Lft Mono Format
//-------------------
if(FxOutputFormat[FxProcessParamSet] == 4)
    {
    TempBuffSize = FxProcessBuffSize;
    TempDataPtr = FxProcessBuffPtr;

    while(TRUE)
        {
        // Store Left Value To Right Channel
        //---------------------------------
        TempValue = *(PINT)(TempDataPtr);
        *(PINT)(TempDataPtr + 4) = TempValue;
        TempDataPtr = TempDataPtr + 8;
        TempBuffSize = TempBuffSize - 8;

        if(TempBuffSize <= 0)
            break;
        }
    }

// Do Rgt Mono Format
//-------------------
if(FxOutputFormat[FxProcessParamSet] == 5)
    {
    TempBuffSize = FxProcessBuffSize;
    TempDataPtr = FxProcessBuffPtr;

    while(TRUE)
        {
        // Store Right Value To Left Channel
        //---------------------------------
        TempValue = *(PINT)(TempDataPtr + 4);
        *(PINT)(TempDataPtr) = TempValue;

        TempDataPtr = TempDataPtr + 8;
        TempBuffSize = TempBuffSize - 8;

        if(TempBuffSize <= 0)
            break;
        }
    }

// Collect Meter Peak Data For Synchronized Display
//------------------------------------------------
TempBuffSize = FxProcessBuffSize;
TempDataPtr = FxProcessBuffPtr;
TempPeakValueL = 0;
TempPeakValueR = 0;

while(TRUE)
    {
    // Get Left Value As Absolute Positive Value
    //-----------------------------------------
    TempValue = *(PINT)(TempDataPtr);

    if(TempValue < 0)
        TempValue = 0 - TempValue;

    // Store As The Highest Peak Value If Needed
    //-----------------------------------------
    if(TempValue > TempPeakValueL)
        TempPeakValueL = TempValue;

    // Get Right Value As Absolute Positive Value
    //------------------------------------------
    TempValue = *(PINT)(TempDataPtr + 4);
```

```
if(TempValue < 0)
    TempValue = 0 - TempValue;

// Store As The Highest Peak Value If Needed
//-------------------------------------------
if(TempValue > TempPeakValueR)
    TempPeakValueR = TempValue;

TempDataPtr = TempDataPtr + 8;
TempBuffSize = TempBuffSize - 8;

// At End Of Buffer Store Final Results For Later Display
//--------------------------------------------------------
if(TempBuffSize <= 0)
    {
    // Trim Value To Max Clip Value If Needed
    //---------------------------------------
    if(TempPeakValueL > 0x007fffff)
        TempPeakValueL = 0x007fffff;

    if(TempPeakValueR > 0x007fffff)
        TempPeakValueR = 0x007fffff;

    // Scale The Peak Value To A Meter Segment Value
    //----------------------------------------------
    for(TempIndex = 0; TempIndex < MAX_METER_SEGMENTS; TempIndex++)
        {
        if(TempPeakValueL <= MeterTable[TempIndex])
            {
            if(TempIndex > 0)
                TempIndex--;

            TempPeakValueL = TempIndex;
            break;
            }
        }

    for(TempIndex = 0; TempIndex < MAX_METER_SEGMENTS; TempIndex++)
        {
        if(TempPeakValueR <= MeterTable[TempIndex])
            {
            if(TempIndex > 0)
                TempIndex--;

            TempPeakValueR = TempIndex;
            break;
            }
        }

    // Simulate Analog Decay For Values Smaller Than The Last Values
    //--------------------------------------------------------------
    if(TempPeakValueL < PlugInMeterLastValueL[FxProcessParamSet] - PlugInMeterDecayFactor[FxProcessParamSet])
        TempPeakValueL = PlugInMeterLastValueL[FxProcessParamSet] - PlugInMeterDecayFactor[FxProcessParamSet];
        if(TempPeakValueR < PlugInMeterLastValueR[FxProcessParamSet] - PlugInMeterDecayFactor[FxProcessParamSet])
            TempPeakValueR = PlugInMeterLastValueR[FxProcessParamSet] - PlugInMeterDecayFactor[FxProcessParamSet];

        // Save These Values For Next Time
        //--------------------------------
        PlugInMeterLastValueL[FxProcessParamSet] = TempPeakValueL;
        PlugInMeterLastValueR[FxProcessParamSet] = TempPeakValueR;

        // Store The Results At The Index Head Position In The Meter Array
        //----------------------------------------------------------------
        TempIndex = PlugInMeterIndexHead[FxProcessParamSet];

        PlugInMeterPos[FxProcessParamSet][TempIndex] = FxProcessPos;
        PlugInMeterValueL[FxProcessParamSet][TempIndex] = TempPeakValueL;
        PlugInMeterValueR[FxProcessParamSet][TempIndex] = TempPeakValueR;

        // Increment Index Head Counter And Set Next Array Value To End Of List Value
        //---------------------------------------------------------------------------
        TempIndex++;

        if(TempIndex >= MAX_METER_INDEXES)
            TempIndex = 0;

        PlugInMeterIndexHead[FxProcessParamSet] = TempIndex;

        PlugInMeterPos[FxProcessParamSet][TempIndex] = END_OF_LIST;
        break;
        }
    }

return;
}
```

**This function does the actual buffer processing on the 16 bit data streams passed to the plug-in by the host program. This routine is called for 16 bit Final Res patches. The data format is 2 bytes per sample per channel which equates to 4 bytes total per sample. This routine functions identical to the 32 bit processing function except for the type casting which handles the 16 bit words instead of the 32 bit dwords.**

```
//===================================
// Do PlugIn Process 16 Bit Stereo
//
// 2 Bytes Per Sample Per Channel
//===================================

void    __stdcall   DoPlugInProcess16(void)

{
int     TempBuffSize;
int     TempValue;
int     TempValueL;
int     TempValueR;
int     TempPeakValueL;
int     TempPeakValueR;
DWORD   TempIndex;
LPBYTE  TempDataPtr;

// Skip If There Is Nothing To Process
//------------------------------------
if(FxProcessBuffSize == 0)
    return;

// Do Volume Process
//------------------
TempBuffSize = FxProcessBuffSize;
TempDataPtr = FxProcessBuffPtr;

while(TRUE)
    {
    // Adjust Value Of Each Sample
    //----------------------------
    TempValue = (int)*(PSHORT)(TempDataPtr);
    TempValue = (TempValue * FxVolFactor[FxProcessParamSet]) / 100;
    *(PSHORT)(TempDataPtr) = (short)TempValue;

    TempValue = (int)*(PSHORT)(TempDataPtr + 2);
    TempValue = (TempValue * FxVolFactor[FxProcessParamSet]) / 100;
    *(PSHORT)(TempDataPtr + 2) = (short)TempValue;

    TempDataPtr = TempDataPtr + 4;
    TempBuffSize = TempBuffSize - 4;

    if(TempBuffSize <= 0)
        break;
    }

// Do Mono Format
//----------------
if(FxOutputFormat[FxProcessParamSet] == 1)
    {
    TempBuffSize = FxProcessBuffSize;
    TempDataPtr = FxProcessBuffPtr;

    while(TRUE)
        {
        // Adjust Left Value Down 6db
        //----------------------------
        TempValueL = (int)*(PSHORT)(TempDataPtr);
        TempValueL = TempValueL / 2;

        // Adjust Right Value Down 6db
        //-----------------------------
        TempValueR = (int)*(PSHORT)(TempDataPtr + 2);
        TempValueR = TempValueR / 2;

        // Add Them Together And Store The Result
        //----------------------------------------
        TempValue = TempValueL + TempValueR;
        *(PSHORT)(TempDataPtr) = (short)TempValue;
        *(PSHORT)(TempDataPtr + 2) = (short)TempValue;

        TempDataPtr = TempDataPtr + 4;
        TempBuffSize = TempBuffSize - 4;

        if(TempBuffSize <= 0)
            break;
        }
    }

// Do Lft Only Format
//--------------------
if(FxOutputFormat[FxProcessParamSet] == 2)
    {
    TempBuffSize = FxProcessBuffSize;
    TempDataPtr = FxProcessBuffPtr;

    while(TRUE)
        {
        // Clear All Right Channel Samples
        //--------------------------------
        *(PSHORT)(TempDataPtr + 2) = 0;

        TempDataPtr = TempDataPtr + 4;
        TempBuffSize = TempBuffSize - 4;

        if(TempBuffSize <= 0)
            break;
        }
    }

// Do Rgt Only Format
//--------------------
if(FxOutputFormat[FxProcessParamSet] == 3)
    {
    TempBuffSize = FxProcessBuffSize;
    TempDataPtr = FxProcessBuffPtr;

    while(TRUE)
        {
        // Clear All Left Channel Samples
        //-------------------------------
        *(PSHORT)(TempDataPtr) = 0;

        TempDataPtr = TempDataPtr + 4;
        TempBuffSize = TempBuffSize - 4;

        if(TempBuffSize <= 0)
            break;
        }
    }
```

```
// Do Lft Mono Format
//-------------------
if(FxOutputFormat[FxProcessParamSet] == 4)
    {
    TempBuffSize = FxProcessBuffSize;
    TempDataPtr = FxProcessBuffPtr;

    while(TRUE)
        {
        // Store Left Value To Right Channel
        //---------------------------------
        TempValue = (int)*(PSHORT)(TempDataPtr);
        *(PSHORT)(TempDataPtr + 2) = (short)TempValue;

        TempDataPtr = TempDataPtr + 4;
        TempBuffSize = TempBuffSize - 4;

        if(TempBuffSize <= 0)
            break;
        }
    }

// Do Rgt Mono Format
//-------------------
if(FxOutputFormat[FxProcessParamSet] == 5)
    {
    TempBuffSize = FxProcessBuffSize;
    TempDataPtr = FxProcessBuffPtr;

    while(TRUE)
        {
        // Store Right Value To Left Channel
        //---------------------------------
        TempValue = (int)*(PSHORT)(TempDataPtr + 2);
        *(PSHORT)(TempDataPtr) = (short)TempValue;

        TempDataPtr = TempDataPtr + 4;
        TempBuffSize = TempBuffSize - 4;

        if(TempBuffSize <= 0)
            break;
        }
    }

// Collect Meter Peak Data For Synchronized Display
//-----------------------------------------------
TempBuffSize = FxProcessBuffSize;
TempDataPtr = FxProcessBuffPtr;
TempPeakValueL = 0;
TempPeakValueR = 0;

while(TRUE)
    {
    // Get Left Value As Absolute Positive Value
```

```
//------------------------------------------
TempValue = (int)*(PSHORT)(TempDataPtr);

TempValue = TempValue << 8;

if(TempValue < 0)
    TempValue = 0 - TempValue;

// Store As The Highest Peak Value If Needed
//------------------------------------------
if(TempValue > TempPeakValueL)
    TempPeakValueL = TempValue;

// Get Right Value As Absolute Positive Value
//-------------------------------------------
TempValue = (int)*(PSHORT)(TempDataPtr + 2);

TempValue = TempValue << 8;

if(TempValue < 0)
    TempValue = 0 - TempValue;

// Store As The Highest Peak Value If Needed
//------------------------------------------
if(TempValue > TempPeakValueR)
    TempPeakValueR = TempValue;

TempDataPtr = TempDataPtr + 4;
TempBuffSize = TempBuffSize - 4;

// At End Of Buffer Store Final Results For Later Display
//-------------------------------------------------------
if(TempBuffSize <= 0)
    {
    // Trim Value To Max Clip Value If Needed
    //---------------------------------------
    if(TempPeakValueL > 0x007fffff)
        TempPeakValueL = 0x007fffff;

    if(TempPeakValueR > 0x007fffff)
        TempPeakValueR = 0x007fffff;

    // Scale The Peak Value To A Meter Segment Value
    //----------------------------------------------
    for(TempIndex = 0; TempIndex < MAX_METER_SEGMENTS; TempIndex++)
        {
        if(TempPeakValueL <= MeterTable[TempIndex])
            {
            if(TempIndex > 0)
                TempIndex--;

            TempPeakValueL = TempIndex;
            break;
            }
```

```
                }

            for(TempIndex = 0; TempIndex < MAX_METER_SEGMENTS; TempIndex++)
                {
                if(TempPeakValueR <= MeterTable[TempIndex])
                    {
                    if(TempIndex > 0)
                        TempIndex--;

                    TempPeakValueR = TempIndex;
                    break;
                    }
                }

            // Simulate Analog Decay For Values Smaller Than The Last Values
            //--------------------------------------------------------------
            if(TempPeakValueL < PlugInMeterLastValueL[FxProcessParamSet] - PlugInMeterDecayFactor[FxProcessParamSet])
                TempPeakValueL = PlugInMeterLastValueL[FxProcessParamSet] - PlugInMeterDecayFactor[FxProcessParamSet];

            if(TempPeakValueR < PlugInMeterLastValueR[FxProcessParamSet] - PlugInMeterDecayFactor[FxProcessParamSet])
                TempPeakValueR = PlugInMeterLastValueR[FxProcessParamSet] - PlugInMeterDecayFactor[FxProcessParamSet];

            // Save These Values For Next Time
            //--------------------------------
            PlugInMeterLastValueL[FxProcessParamSet] = TempPeakValueL;
            PlugInMeterLastValueR[FxProcessParamSet] = TempPeakValueR;

            // Store The Results At The Index Head Position In The Meter Array
            //----------------------------------------------------------------
            TempIndex = PlugInMeterIndexHead[FxProcessParamSet];

            PlugInMeterPos[FxProcessParamSet][TempIndex] = FxProcessPos;
            PlugInMeterValueL[FxProcessParamSet][TempIndex] = TempPeakValueL;
            PlugInMeterValueR[FxProcessParamSet][TempIndex] = TempPeakValueR;

            // Increment Index Head Counter And Set Next Array Value To End Of List Value
            //---------------------------------------------------------------------------
            TempIndex++;

            if(TempIndex >= MAX_METER_INDEXES)
                TempIndex = 0;

            PlugInMeterIndexHead[FxProcessParamSet] = TempIndex;

            PlugInMeterPos[FxProcessParamSet][TempIndex] = END_OF_LIST;
            break;
            }
        }

    return;
    }
```

**This function resets the meter display for a specific parameter set by refreshing the meter bitmap rectangle.**

```
//=====================
// Reset Meter Display
//=====================

void    __stdcall   ResetMeterDisplay(DWORD TempParamSet)

{
int     TempSrcXPos;
int     TempSrcYPos;
int     TempDestXPos;
int     TempDestYPos;
int     TempDestWidth;
int     TempDestHeight;
HDC     TempMemoryDC;

if(PlugInParamSetTable[TempParamSet] == 0)
    return;

if(hWndPlugIn[TempParamSet] == 0)
    return;

// Create Compatible Memory DC
//----------------------------
TempMemoryDC = CreateCompatibleDC(PlugInDC[TempParamSet]);

// Clear Lft And Rgt Meter
//------------------------
SelectObject(TempMemoryDC, hPlugInMainBitmap);

TempDestXPos = RectPlugInMeterPeakL.left;
TempDestYPos = RectPlugInMeterPeakL.top;
TempDestWidth = RectPlugInMeterR.right - RectPlugInMeterL.left + 1;
TempDestHeight = RectPlugInMeterL.bottom - RectPlugInMeterPeakL.top + 1;

TempSrcXPos = RectPlugInMeterPeakL.left;
TempSrcYPos = RectPlugInMeterPeakL.top;

BitBlt(PlugInDC[TempParamSet], TempDestXPos, TempDestYPos, TempDestWidth, TempDestHeight, TempMemoryDC, TempSrcXPos, TempSrcYPos, SRCCOPY);

DeleteDC(TempMemoryDC);
return;
}
```

**This function displays all active meter displays by referencing the meter variable arrays at the tail count position. The current sample position is compared to the next position value in the array and when the positions match, the data is collected and displayed. The meter is drawn from the bottom pixel to the segment height of the current value, then the top portion of the meter rectangle is cleared. Note that the position is checked in a loop fashion to make sure to come up to the last array data that is closest to the current sample position. This keeps the meter display synchronized with the playback at all times.**

```
//====================
// Display All Meters
//===================

void    __stdcall   DisplayAllMeters(void)

{
DWORD   TempSet;
int     TempPeakValueL;
int     TempPeakValueR;
DWORD   TempIndex;
int     TempSrcXPos;
int     TempSrcYPos;
int     TempDestXPos;
int     TempDestYPos;
int     TempDestWidth;
int     TempDestHeight;
HDC     TempMemoryDC;

// Skip If There Is Nothing To Display
//-----------------------------------
if(PlugInParamSetCount == 0)
    return;

// Only Display Meters During Real-time Playback
//---------------------------------------------
if(FxProcessFlag != REALTIME_PLAYBACK)
    return;

TempMemoryDC = 0;

// Display All Active And Visible Parameter Set Meters
//----------------------------------------------------
for(TempSet = 0; TempSet < MAX_PARAM_SETS; TempSet++)
    {
    // Skip If Not Active
    //------------------
    if(PlugInParamSetTable[TempSet] == 0)
        continue;

    // Skip If Not Visible
    //-------------------
    if(hWndPlugIn[TempSet] == 0)
        continue;

    // Skip If Bypassed
    //----------------
    if(PlugInDataValue[TempSet][AUTO_BYPASS_SW])
        continue;

    // Find The Next Entry That Is Closest To The Current Playback Sample Position
    //---------------------------------------------------------------------------
    LoopBack:
    // Skip If The Head And Tail Index Counters Are The Same
    //------------------------------------------------------
```

```
if(PlugInMeterIndexHead[TempSet] == PlugInMeterIndexTail[TempSet])
    continue;

// Start Looking From The Index Tail Counter
//-----------------------------------------
TempIndex = PlugInMeterIndexTail[TempSet];

// Skip If Current Position Has Not Yet Reached Next Entry Position
//----------------------------------------------------------------
if(FxSamplePos < PlugInMeterPos[TempSet][TempIndex])
    continue;

// Get Values
//-----------
TempPeakValueL = PlugInMeterValueL[TempSet][TempIndex];
TempPeakValueR = PlugInMeterValueR[TempSet][TempIndex];

// Increment Index Tail Counter
//----------------------------
TempIndex++;

if(TempIndex >= MAX_METER_INDEXES)
    TempIndex = 0;

PlugInMeterIndexTail[TempSet] = TempIndex;

// Loopback If Current Position Is Greater Than Next Position
//----------------------------------------------------------
if(FxSamplePos >= PlugInMeterPos[TempSet][TempIndex])
    goto LoopBack;

// Create Compatible Memory DC If Needed
//-------------------------------------
if(TempMemoryDC == 0)
    TempMemoryDC = CreateCompatibleDC(PlugInDC[TempSet]);

// Fill Lft Meter From Bottom To Cur Peak
//--------------------------------------
if(TempPeakValueL)
    {
    SelectObject(TempMemoryDC, hPlugInMeterBitmap);

    TempDestXPos = RectPlugInMeterL.left;
    TempDestYPos = RectPlugInMeterL.bottom - TempPeakValueL;
    TempDestWidth = PlugInMeterBitmapHdr.bmWidth;
    TempDestHeight = TempPeakValueL;

    TempSrcXPos = 0;
    TempSrcYPos = PlugInMeterBitmapHdr.bmHeight - TempPeakValueL;

    BitBlt(PlugInDC[TempSet], TempDestXPos, TempDestYPos, TempDestWidth, TempDestHeight, TempMemoryDC, TempSrcXPos,
          TempSrcYPos, SRCCOPY);
    }
```

```
// Clear Rest Of Lft Meter To Top
//--------------------------------
TempDestHeight = (RectPlugInMeterL.bottom - RectPlugInMeterL.top + 1) - TempPeakValueL;

if(TempDestHeight)
    {
    SelectObject(TempMemoryDC, hPlugInMainBitmap);

    TempDestXPos = RectPlugInMeterL.left;
    TempDestYPos = RectPlugInMeterL.top;
    TempDestWidth = PlugInMeterBitmapHdr.bmWidth;

    TempSrcXPos = RectPlugInMeterL.left;
    TempSrcYPos = RectPlugInMeterL.top;

    BitBlt(PlugInDC[TempSet], TempDestXPos, TempDestYPos, TempDestWidth, TempDestHeight, TempMemoryDC, TempSrcXPos,
          TempSrcYPos, SRCCOPY);
    }

// Fill Lft Meter Peak If Needed
//-------------------------------
if(TempPeakValueL >= MAX_METER_SEGMENTS - 2)
    {
    SelectObject(TempMemoryDC, hPlugInMeterBitmap);

    TempDestXPos = RectPlugInMeterPeakL.left;
    TempDestYPos = RectPlugInMeterPeakL.top;
    TempDestWidth = PlugInMeterBitmapHdr.bmWidth;
    TempDestHeight = RectPlugInMeterPeakL.bottom - RectPlugInMeterPeakL.top + 1;

    TempSrcXPos = 0;
    TempSrcYPos = 0;

    BitBlt(PlugInDC[TempSet], TempDestXPos, TempDestYPos, TempDestWidth, TempDestHeight, TempMemoryDC, TempSrcXPos,
          TempSrcYPos, SRCCOPY);
    }

// Fill Rgt Meter From Bottom To Cur Peak
//----------------------------------------
if(TempPeakValueR)
    {
    SelectObject(TempMemoryDC, hPlugInMeterBitmap);

    TempDestXPos = RectPlugInMeterR.left;
    TempDestYPos = RectPlugInMeterR.bottom - TempPeakValueR;
    TempDestWidth = PlugInMeterBitmapHdr.bmWidth;
    TempDestHeight = TempPeakValueR;

    TempSrcXPos = 0;
    TempSrcYPos = PlugInMeterBitmapHdr.bmHeight - TempPeakValueR;

    BitBlt(PlugInDC[TempSet], TempDestXPos, TempDestYPos, TempDestWidth, TempDestHeight, TempMemoryDC, TempSrcXPos,
          TempSrcYPos, SRCCOPY);
    }
    // Clear Rest Of Lft Meter To Top
    //-------------------------------
    TempDestHeight = (RectPlugInMeterR.bottom - RectPlugInMeterR.top + 1) - TempPeakValueR;

    if(TempDestHeight)
        {
        SelectObject(TempMemoryDC, hPlugInMainBitmap);

        TempDestXPos = RectPlugInMeterR.left;
        TempDestYPos = RectPlugInMeterR.top;
        TempDestWidth = PlugInMeterBitmapHdr.bmWidth;

        TempSrcXPos = RectPlugInMeterR.left;
        TempSrcYPos = RectPlugInMeterR.top;

        BitBlt(PlugInDC[TempSet], TempDestXPos, TempDestYPos, TempDestWidth, TempDestHeight, TempMemoryDC, TempSrcXPos,
              TempSrcYPos, SRCCOPY);
        }

    // Fill Rgt Meter Peak If Needed
    //------------------------------
    if(TempPeakValueR >= MAX_METER_SEGMENTS - 2)
        {
        SelectObject(TempMemoryDC, hPlugInMeterBitmap);

        TempDestXPos = RectPlugInMeterPeakR.left;
        TempDestYPos = RectPlugInMeterPeakR.top;
        TempDestWidth = PlugInMeterBitmapHdr.bmWidth;
        TempDestHeight = RectPlugInMeterPeakR.bottom - RectPlugInMeterPeakR.top + 1;

        TempSrcXPos = 0;
        TempSrcYPos = 0;

        BitBlt(PlugInDC[TempSet], TempDestXPos, TempDestYPos, TempDestWidth, TempDestHeight, TempMemoryDC, TempSrcXPos,
              TempSrcYPos, SRCCOPY);
        }
    }

if(TempMemoryDC)
    DeleteDC(TempMemoryDC);

return;
}

//----
```