XRS®

Extensible Radio Specification

Version 1.2

CONFIDENTIAL

Limited Evaluation License

WiNRADiO Communications (hereinafter called 'WiNRADiO') hereby grants to you at no charge a non-exclusive, non-transferable, worldwide, limited license (without any right to sublicense) under WiNRADiO's intellectual property rights that are essential to practice the Extensible Radio Specification (XRS, hereinafter called 'Specification') to use the Specification for internal evaluation purposes only. Other than this limited license, you acquire no right, title, or interest in or to the Specification and you shall have no right to use the Specification for productive or commercial use. Should you wish to use the Specification for productive or commercial use, you need to apply for XRS Client Developer License or XRS Server OEM License.

Confidential Information

This documentation is the confidential and proprietary information of WiNRADiO. You shall not disclose this confidential information and shall use it only in accordance with the terms of this license agreement you entered into with WiNRADiO.

Limited Warranty

The specification is provided 'AS IS' and all warranties whether express, implied, statutory or otherwise, relating in any way to the subject matter of this licence or to this licence generally, including without limitation, warranties as to: quality; fitness; merchantability; correctness; accuracy; reliability; correspondence with any description or sample, meeting your or any other requirements; uninterrupted use; compliance with any relevant legislation and being error or virus free are excluded.

Where any legislation implies in this agreement any term, and that legislation avoids or prohibits provisions in a contract excluding or modifying such a term, such term shall be deemed to be included in this agreement. However, the liability of WiNRADiO for any breach of such term shall if permitted by that legislation be limited, at WiNRADiO's option, to any one or more of the following: if the breach related to goods: the replacement of the goods or the supply of equivalent goods; the repair of such goods; the payment of the cost of replacing the goods or of acquiring equivalent goods; or the payment of the cost of having the goods repaired; and if the breach relates to services: the supplying of the services again; or the payment of the cost of having the services supplied again.

Licensee warrants that it has not relied on any representation made by WiNRADiO or upon this specification in any way.

Limitation of Liability

Except for the limited warranty (above), WiNRADiO shall not be under any liability to Licensee in respect of any loss or damage (including consequential loss or damage) however caused, which may be suffered or incurred or which may arise directly or indirectly in respect to the use of this specification or the failure or omission on the part of WiNRADiO to comply with its obligations under this licence.

Trademarks

WiNRADiO, VisiTune and XRS are trademarks or registered trademarks of WiNRADiO Communications in the United States of America and other countries of the World.

Copyrights

XRS 1.2

Contents

Introduction

The XRS (Extensible Radio Specification) is a standard-based platform for the control of radio devices (such as receivers or transmitters) by a computer.

The need for this platform has arisen due to the increased integration of radio and computing devices. Many new radio communications protocols (for example trunking radio) and modulation modes (such as Digital Audio Broadcasting) require the availability of computing power. At the same time, computing power is being used to enhance radio devices by adding extra functions and, increasingly often, to replace conventional front panel controls by a computer keyboard and screen.

A standard to facilitate a uniform way of controlling radio devices by a computer has been long overdue.

The XRS defines the interface between a radio device control program (the 'Server') and an add-on plug-in module (the 'Client'). This specification is flexible enough to allow for a wide range of radio devices to be controlled by a wide range of such plug-ins. This means that application software developed for one particular model of a radio device will work equally well on another, provided both the radio device software and the plug-in are XRS-compliant.

The introduction of XRS benefits everyone:

- End users, because XRS application software purchased for one model of an XRS-compliant radio device will work equally well with another. The greater number of XRS applications, the better value an investment in an XRS-compliant radio device will represent for the radio device user.
- Developers, because once developed, an XRS application software will work across all XRS-compliant radio devices, not just one model of one particular manufacturer. This saves time for application development and increases the market size for radio device applications, thus providing a better return on investment and a greater incentive to develop add-on software.
- Manufacturers, because they can take advantage of already existing XRS applications. By making a radio device software XRS-compatible, manufacturers make it automatically more useful to prospective purchasers, and therefore more attractive to the market.

The primary design goals for XRS were:

- To enable new radio functions to be developed separately, in a modular way, and added quickly and easily to expand the functionality of the radio device control software.
- To eliminate compatibility issues between different models of radio devices. Once written, an XRS application will work with every XRS-compliant radio device.
- To provide an open platform for third party software developers. The XRS developer information exposes all interfaces needed to enable development of new radio control tools and applications. The XRS standard is designed from the ground up to provide a flexible platform for development of software suitable for a wide variety of radio applications.
- To provide the opportunity for radio device manufacturers to take advantage of existing and future XRS applications. By licensing the XRS server technology for use in their products, other radio manufacturers will benefit of the combined efforts of all third-party XRS developers.
- To ensure that the XRS standard itself is extensible in order to be able to accommodate new advances in both radio and computer technology.

The XRS API as described in this document is provided for evaluation only, under the terms of the Evaluation License Agreement with which you agreed. It cannot be used for any other productive or commercial purpose.

If you wish to become an XRS Client developer, you need to apply for the XRS Developer's License. This license imposes certain essential restrictions and requirements, which are necessary to protect the integrity of the XRS specification. This license, when granted, is free. You may download the Developer's License Agreement from the XRS Web page: **http://xrs.winradio.com**. Upon signing of the License Agreement by both parties, you will be also given access to a special XRS Resource Page, with source code examples that substantially simplify XRS Client development.

If you wish to develop XRS Server software which controls the radio device at the lowest level (ie. such software which is capable of accepting XRS plug-ins), you will need to apply for an XRS OEM License. This license is not free. You may also apply for such license from the above XRS Developer's Web page.

XRS software can be developed in most major development environments including C/C++ and Delphi. An XRS implementation is platform-specific and therefore must be ported to every operating system and processor platform upon which it is to be deployed.

This document refers mainly to the Windows environment but in the future will expand to cover Macintosh and Linux (parts of this document will outline differences between the three platforms).

Note: *In the following text, a 'plug-in' or a 'client' will mean 'XRS Client', and 'server' will mean 'XRS Server', unless indicated otherwise.*

XRS Plug-in Basics

How Plug-ins Work

When an XRS-compliant radio device application ('XRS Server') starts, it searches for plug-in files with a '.XRS' extension in the `plugins` folder in the same folder as the application. To allow plug-ins to share the same location for different applications, the following locations can be checked:

- 32-bit Windows: the application should check the registry for the `SearchPath` value in the `HKEY_LOCAL_MACHINE\Software\XRS` key. Plug-ins should store setting information in a subkey in the `HKEY_CURRENT_USER\Software\XRS` key.
- Linux and 16-bit Windows: the application should check the path set by the optional environment variable XRS_PLUGIN_PATH.

When the application starts it searches the folder(s) for all XRS plug-ins and attempts to load and initialise all plug-ins that are found.

The following stages outline the life of a plug-in from loading to shutdown:

- At start-up, the plug-in is loaded and the `xrsPluginInit` function is called to inform the plug-in of the supported XRS version. The plug-in returns the name of itself and informs the application what it does. It can also inform the application to start it immediately.
- When the plug-in is to be started (by auto-starting, user initiation or by another plug-in), the `xrsPluginStart` function is called. A new instance of the plug-in has to be created and the plug-in instance has to return a unique identifier to itself. Multiple instances can exist when more than one radio device is open in the same application where each device has started the same plug-in. Each radio device instance passes a unique identifier to the plug-in instance when it is started.
- After a plug-in instance is started, it will receive notifications (or events) from the application through the `xrsPluginNotify` function. It can also control most aspects of the application if it wishes to do so.
- When a plug-in instance receives a `PN_CLOSE` notification, it must shutdown and destroy the instance that was started. The plug-in instance must inform the application that it has closed down by issuing a `PM_CLOSED` command.
- When the application closes down, the `xrsPluginDone` function is called to let the plug-in know it is about to be unloaded from memory.

Note: *XRS API calls and callbacks use the main application thread. In general, if a plug-in intends to generate additional threads to handle processing at any stage in its lifespan, care should be taken to isolate these from the API calls.*

Overview of Plug-in Structure

A plug-in is a native code library whose file extension is .XRS. Internally, the file type depends on the platform:

- *Windows:* Dynamic Link Library files (.DLL)
- *Linux:* Shared Object files (.SO or .DSO)
- *Mac OS:* PowerPC Shared Library files or 68K code resources

The actual programming language used does not matter as long as it can generate at least one of the above file types.

Although it is a native code library and is therefore platform specific and runs from an XRS compliant application, the XRS API is designed to provide the maximum degree of flexibility and to be functionally consistent across all platforms. The main platform specific differences will occur from the underlying operating system's API.

Note: *XRS is different from other platform-native inter-application architectures such as OLE, where components developed for these systems are relatively complex and heavyweight. XRS is specifically designed to extend radio communications software and are therefore relatively simple and lightweight.*

XRS Development Overview

Conventions

Throughout the rest of the document, the following terms are used:

DSP:	any device performing analog-digital and/or digital-analog conversion, and may include an either digital signal processor in between, or DSP emulation in software
DWORD:	a 32-bit integer (typically signed unless stated)
High word:	high 16-bits of a DWORD (bits 16 – 31), typically signed unless stated
Low word:	low 16-bits of a DWORD (bits 0 – 15), typically signed unless stated
NULL:	zero
XRS:	Extensible Radio Specification

Writing Plug-ins

Once you become a Licensed XRS Client Developer, creating XRS plug-ins is a simple process:

1. You can derive your plug-in from sample templates provided on the XRS Client Developer Resource Web Page or you can construct a plug-in from scratch using header files.
2. Each plug-in requires the implementation of four essential API functions that are documented in the following pages. The plug-in must be able to close when requested to do so by the server.
3. Once a plug-in has been written, it must be built and installed.
4. Finally your plug-in has to be tested and debugged as necessary.

Note: *You can avoid many developmental problems by working in stages and testing at each stage. In the first stage, you can create a plug-in project, handle the basic functionality first (described later), build it and install it. In the second stage, you add the special functionality that makes the plug-in unique. The following chapters will give more information about the functionality that can be added.*

Registering Plug-ins

In order to identify a plug-in, XRS-compliant applications (XRS servers) first locate the plug-in library file with a .XRS extension and load it into memory.

For each plug-in loaded, the application calls the `xrsPluginInit` function to inform the plug-in which version of XRS it supports. If the plug-in handles the XRS version correctly, and if the plug-in verifies the validity of the XRS server, it then returns its name (up to 64 characters in length) and what type of plug-in it is (there are several broad categories defined). The application generally puts all supported plug-ins in a menu for the user to initiate. Each plug-in should have a unique name to avoid confusion for the user.

The return value also specifies the degree of control the plug-in can exercise over the server, and how the server should handle it. For example, the plug-in can request the server to start it immediately (an 'auto-start' plug-in), to hide it from the menu (a 'hidden' plug-in, which can be only started automatically or from another plug-in), etc.

Note: *An auto-start plug-in can also 'hide' the server and so effectively take over the user interface of the radio device.*

Initialisation

The application calls `xrsPluginInit` once when the plug-in is loaded, before the first instance is created. This function is used to allocate the memory and resources shared by all instances of an XRS plug-in.

C/C++:

```
int xrsPluginInit(int iXRSVer, PCHAR lpServerId, PCHAR lpName,
   int cbName);
```

Delphi:

```
function xrsPluginInit(iXRSVer: Integer; lpServerId, lpName: PChar;
   cbName: Integer): Integer;
```

When the application shuts down, the application calls `xrsPluginDone`, which releases the memory or resources allocated by `xrsPluginInit`.

The plug-in should check the version information in the `iXRSVer` parameter to verify that it is compatible with the API capabilities provided by the application. The low-byte specifies the minor version number, and next most significant specifies the major version number. For example, for version 1.0 `iXRSVer` will equal 0x0100.

Next the plug-in must validate the server ID by calling `xrsValidateServer` and if it is valid, the function returns TRUE. If `xrsValidateServer` returns false, the plug-in must return zero.

No XRS API calls can take place in either direction until the initialisation completes successfully.

If a plug-in cannot initialise (resource allocation error, unsupported XRS version, or invalid XRS server), it must return zero.

Successful initialisation must return a positive value, where the lowest byte informs the server about the degree of control this plug-in is allowed:

Bits 0 & 1:

0 = Failure

1 = Doesn't take control, but can occasionally change various settings.

2 = Takes temporary control, it should disable controls that is overriding.

3 = Takes full control, hides/minimises/disables panel or appropriate controls.

Bits 2 & 3:

0 = Doesn't use DSP.

4 = Occasionally uses the DSP, it will have to share with other similar plug-ins.

8 = DSP read raw audio data, can operate with other plug-ins that also use the same data.

12 = DSP full control, no other DSP based plug-ins can operate at the same time.

Note: *'DSP' means the DSP facility available as part of the radio device. This does not include the signal processing facility (sound card) provided by the control computer.*

Bit 4:

Uses a sound device, cannot operate with other similar plug-ins.

Note: *'Sound device' is typically a sound card available on the control computer.*

Bit 5:

Informs the application to start the plug-in immediately. (An 'auto-start' plug-in.)

Bit 6:

Informs the application not to show the plug-in in the plug-in list and/or menu but can still be accessed from the `PM_GETNEXTPLUGIN` command. (A 'hidden' plug-in.)

Bit 7:

Indicates that the plugin window handle will be returned in xrsPluginStart function and allows the XRS server to manipulate it (for example to embed to the application panel).

The second lowest byte specifies the class of the plug-in (this can be ORed with the above bit specifiers):

0x0000: Standard class – most plug-ins will use this category.

0x0100: Trunking class – plug-ins that perform trunking operations (decoding, tracking, etc) use this category.

0x0200: DSP class – plug-ins that perform digital signal processing use this category.

0x0300: Decoder class – plug-ins that perform any decoding of the received signal use this category.

0x0400: Demodulator class - plugins that perform demodulating of the received signal use this category. For proper running of all other plug-ins, demodulator class plug-ins should be the last ones destroyed.

0x0500: DF class - plugins that perform direction finding of the received signal use this category.

Note: *The class specifiers may be for example used for the plug-ins to appear in different menus in the server application. For example, in WiNRADiO receivers, Standard class plug-ins are shown in the "Plug-ins" menu, Trunking class in the "Trunking" menu, DSP class in the "Digital Suite" menu, Decoder class in the "Decoders" menu, etc.*

Instance Creation

Any time after initialisation, a plug-in instance can be created or started. More than one instance of the same plug-in can exist if the application supports operation of multiple radio devices or the same application space supports operation of multiple devices.

A plug-in instance can be created by user initiation from a menu, started automatically by the application (at start-up or any time after) or from another plug-in.

C/C++:

```
DWORD xrsPluginStart(HWND hAppWnd, LPRADIODEVCAPS lpRadioDevCaps,
  PLUGINPROC lpPluginProc, PDWORD lpFilterFlags);
```

Delphi:

```
function xrsPluginStart(hAppWnd: HWND; lpRadioDevCaps: PRadioDevCaps;
  lpPluginProc: TPluginProc; var lpFilterFlags: Longint): Integer;
```

The `hAppWnd` parameter specifies a handle to the application's device window. It can either be the application's main window if it only controls one device or it can be child window for the device if the application supports control of multiple devices. Generally, this is just used to identify the radio device in a multiple instance situation.

The `lpRadioDevCaps` parameter points to a `RADIODEVCAPS` structure which specifies the capabilities of the radio device. The radio device can support receiving, transmitting or both. The memory utilised by this

structure is only temporary so if the information is required later, a copy must be made. The plug-in should also check various fields to make sure it can support the device properly and that it is suitable for the device.

The `lpPluginProc` parameter is a pointer to a application provided callback function that the plug-in can use to control the operation of the radio device and also perform other operations including informing the application when it closes down.

C/C++:

```
typedef DWORD (CALLBACK* PLUGINPROC)(DWORD, int, DWORD, int, LPVOID);
```

Delphi:

```
type
  TPluginProc = function (hPlugin: Longint; uMsg: Integer;
     dwParam: Longint; cbData: Integer; lpData: Pointer): Longint;
```

After a `PM_CLOSED` command is issued through this callback, the plug-in will not receive any more notifications nor can it issue any commands.

The `lpFilterFlags` parameter points to a 32-bit variable that the plug-in may use, with a combination of `PNF_xxx` flags, to filter out any notifications that it does not wish to receive. Filter flags are defined for broad categories of notifications, so if one notification is required in a group of other notifications that are not required, then the group should not be filtered. The notification filtering change be changed at any time during the instance of the plug-in by issuing a `PM_FILTERFLAGS` command. By default, no notifications are filtered out (that is, all notifications are received). Any notifications that are not required by the plug-in can be ignored except the `PN_CLOSE` notification.

During the start procedure, a plug-in may issue commands to the application but must pass an instance identifier of zero to the callback procedure.

If the plug-in instance starts successfully, it must return a positive value that is unique for the plug-in instance. The application will not use the value except as an identifier in the notification function.

If it is not started successfully, the function must return zero.

Device Information

After a plug-in has been installed, it can be started at any time. When the plug-in is started, a structure is passed to the plug-in to provide detailed information on the underlying radio device (receiver and/or transmitter). Information includes the manufacturer and product, supported frequency ranges and modes and other capabilities of the device.

Note: *For more information on the information provided, see the section on the XRS Structures, especially the Radio Device Capabilities Structure (`RADIODEVCAPS`).*

Instance Destruction

Any time after an instance is created with `xrsPluginStart`, the instance can be closed with a `PN_CLOSE` notification received in the `xrsPluginNotify` command.

Instances can be closed by the user, another plug-in or when the application shuts down. A plug-in instance can also close itself.

When the instance is closed, it must issue a `PM_CLOSED` command to the application, after which the plug-in will not receive any more notifications nor can it issue any further commands. If the plug-in sets bit 16 in the `dwParam` parameter of the `PM_CLOSED` command, the application's radio device instance should close too. If the application supports only one radio device, the entire application should close.

Shutdown

When the application is about to close, before it unloads the plug-in module, it will call the plug-in's `xrsPluginDone` function. This gives the plug-in an opportunity to delete all data and resources allocated in the `xrsPluginInit` function that were shared by all instances. It also gives the plug-in a change to cancel any outstanding I/O requests, delete threads it created, free memory and perform any other closing tasks.

C/C++:

```
void xrsPluginDone(void);
```

Delphi:

```
procedure xrsPluginDone;
```

Minimal Plug-in Example

This example demonstrates the minimum requirements for a plug-in in C:

```
/* Global variable to store callback address */
PLUGINPROC PluginProc = NULL;
static char PluginName[] = "Minimal Plug-in Example";

/* Initialisation function */
int XRSAPI xrsPluginInit(int iXRSVer, PCHAR lpServerId, PCHAR lpName,
   int cbName)
{
   /* Validate the server */
   if ( !xrsValidateServer( lpServerId ))
         return 0;

   /* Tell application the name of the plug-in */
   strncpy( lpName, PluginName, cbName );

   /* Plug-in initialised correctly, and is a simple plug-in type */
   return 1;
}

#define INSTANCE_HANDLE 1

/* Creation function */
DWORD XRSAPI xrsPluginStart(HWND hAppWnd, LPRADIODEVCAPS
   lpRadioDevCaps, PLUGINPROC lpPluginProc, PDWORD lpFilterFlags)
{
   /* If PluginProc is defined, then an instance is already running */
   if ( PluginProc ) return 0;

   /* Store pointer to callback function */
   PluginProc = lpPluginProc;

   /* Receive no notifications */
   *lpFilterFlags = PNF_ALL;

   /* Since this is a single instance plug-in, return any value */
   return INSTANCE_HANDLE;
}

void XRSAPI xrsPluginNotify(HWND hAppWnd, int uMsg, DWORD dwData,
   int cbData, LPVOID lpData)
{
   /* Must handle close message */
   if ( uMsg == PN_CLOSE )
   {
         /* Inform application that the plug-in has closed */
         PluginProc( INSTANCE_HANDLE, PM_CLOSED, 0, sizeof(PluginName),
               PluginName );

         /* Clear PluginProc, allows another instance to be started */
         PluginProc = NULL;
   }

   /* ignore all other notifications that may arrive */
```

```
}

/* Shutdown function */
void XRSAPI xrsPluginDone(void)
{
   /* Nothing to do here! */
}
```

XRS Event Handling and Control

This chapter deals with specifics on what can be done with plug-ins and how to implement the details.

Start-up Conditions

During and after an `xrsPluginStart` call, the plug-in can start making calls to the application. Generally, during start up, the plug-in will retrieve any settings it will use during the life of the plug-in and initialise any controls with the settings. It could also put the radio device into a particular state for the plug-in to operate in. During the start-up stage (before `xrsPluginStart` returns with an instance handle), any calls to the application must be made with an instance handle (`hPlugin`) of zero.

Notifications will not be issued until `xrsPluginStart` returns and in most cases, only are issued when a setting changes.

Notifications

Notifications fall into several categories:

- Application notifications: Informs the plug-in of any changes to the device's user interface. These include disabling (or taking control) of various functions, hiding and showing of the entire interface and minimisation of the interface.
- Receiver notifications: Informs the plug-in of any changes to the settings of the receiver. These include reception frequency, demodulation mode, signal level, squelch control, IF strip functions, audio functions, etc.
- Transmitter notifications: Informs the plug-in of any changes to the settings of the transmitter. These include input settings, transmission frequency, modulation parameters, output power, etc.
- Plug-in notifications: Informs the plug-in of any other plug-ins that have been started or stopped.
- Memory notifications: Informs the plug-in when any changes are made to the receiver's frequency memory. These include new, modified or deleted entries, bank changes, folder changes and file changes.
- DSP notifications: XRS supports DSP functionality including DAC and ADC on both receiver outputs and transmitter inputs for advanced reception and/or transmission functionality. Functionality can include modulation and demodulation, coding and decoding, recording and playback, etc. Notifications include DSP requests, acknowledgments of transferred data and DSP state changes.

Commands

Commands fall into several categories similar to notifications:

- Application commands: Commands can be issued to the application to control how the device's user interface behaves. These include hiding, showing and minimising the entire interface. A plug-in can also disable various groups of controls so the user cannot alter any settings that the plug-in is controlling.
- Receiver commands: Commands can be sent to change any settings that the receiver (and application) supports. For most notifications that the plug-in receives, there is an equivalent command to change the setting from the plug-in.

The only exceptions are the signal level and status (including squelch, scanning and DSP).

- Transmitter commands: Commands can also be issued to the application to control any aspect of transmitter operation.
- Plug-in commands: Commands can be sent to the application to obtain a list of all installed plug-ins and to find out if they are currently running (notifications are issued to inform all plug-ins if another plug-in is started or stopped). Plug-ins can also control other plug-ins by starting and/or stopping them.
- Memory commands: A plug-in can issue commands to add, modify and delete entries in the receiver's frequency memory. It can also select a different bank or folder (if the application supports it) and load a different memory file. A plug-in can also command the application to recall a frequency from memory.
- DSP commands: XRS supports receiver and transmitter DSPs and/or ADC/DACs independently of each other. If DSP facilities are available on a radio device, XRS plug-ins can be used for a large variety of functions.

User Interface Control

Taking Control of the Device's Functions

When a plug-in wishes to take control of specific functions of the device, the plug-in can disable the controls on the user interface to prevent the user from overriding the plug-ins operation. This same feature also stops other plug-ins from attempting to control the same functions.

A plug-in has three ways of finding out if it can take control of a device's function:

- Monitoring the disable status (from the `PN_DISABLED` notification).
- Obtaining the interface's disabled status (using the `PM_GETSETTINGS` command).
- Trying to disable the functions (using the `PM_DISABLE` command) and checking the return value from the call.

The user interface will disable the controls associated with the flags set in the `PM_DISABLE` command and will block any other plug-ins from attempting to control the disabled features.

Memory Control

XRS has support for comprehensive interaction with the radio device frequency memory. It can recall frequencies, store, modify and delete frequencies from memory. For radio devices that utilise banks, the plug-in can select or obtain the current memory bank. Other radio devices may support frequency storage in a folder system similar to the file system used in most operating systems, plug-ins can read and modify the folder tree as well as select or obtain the active folder.

Reading from Frequency Memory

The plug-in can read the contents of the memory file by first issuing a `PM_GETNEXTMEM` command with a parameter of –1 (this obtains the first memory number) and supplying a `MEMORYENTRY` buffer to be filled by the command. The command will return the number of the first used record (and the buffer will be filled with that record). Repeat the calls to `PM_GETNEXTMEM` passing the return value from the previous call until the command returns –1 (no more memory records). The call can also pass NULL for the buffer (in the `lpData` parameter) to just obtain a list of used memory records.

If the plug-in just wants the number of records, it can issue a `PM_GETNUMMEMS` command. The plug-in can also obtain the contents of a specific memory record by issuing a `PM_GETMEM` command passing the memory record number and supplying a `MEMORYENTRY` buffer to be filled.

To get the application to recall a memory record, the plug-in can issue a `PM_RECALLMEM` command passing the record number. The application will recall the settings in the record and appropriate apply the settings to the receiver. When a memory is recalled, the application will issue a `PN_MEMRECALL` notification to all active plug-ins (unless the notification is filtered out with a `PNF_MEMORY` filter flag).

Modifying Frequency Memory

To add or modify a memory record, the plug-in can issue a PM_STOREMEM command. If the record doesn't exist, a new record is created. If the record does exist, the record will be overwritten with the data supplied by the command.

If the *lpData* parameter is NULL, the memory record is deleted.

Whenever a memory record is added, modified or deleted, the application will issue a PN_MEMCHANGE notification to all running plug-ins. The notification supplies the memory record number and the new contents of the record in a MEMORYENTRY structure (or NULL if the record was deleted).

File Selection

If the application supports different frequency memory files that can be loaded, the plug-in can obtain the currently open memory file with the PM_GETMEMFILE command. The plug-in can also issue a PM_SETMEMFILE command to the application to load a different file. If the application does not support multiple memory file support, these functions will fail.

When a new file is opened or created, a PN_MEMFILE notification will be sent to all active plug-ins including the name of the file opened.

Bank Selection

Many receivers store memory records in separate banks and only one bank can be accessed at one time. If the memory utilises banks, the RADIOMEM_BANKS flag will be set in the *dwMemFeatures* field of the RADIODEVCAPS structure. The *iNumBanks* field specifies the number of banks that the frequency memory has.

Plug-ins can select the active bank by issuing a PM_SELECTBANK command. When the active bank is changed, the application issues a PN_MEMBANK notification to all active plug-ins.

Folder Manipulation

An alternative to banks in some receivers, is a tree structure to store memory records into. Like banks, only one 'folder' can be active at one time, and folders can be nested in other folders. Selecting folders operates in a similar way to changing active directories in a computer's file system. To select the active folder, the plug-in can issue a PM_OPENFOLDER command specifying a relative path to the active folder or a full path. Sub-folders are separated by the back-slash character '\', and folder names can include any character except a back-slash. Generally, folder names should only include standard ASCII characters ranging from 32 (space) to 126 (tilde '~'). To open an absolute folder path, precede the path with a back-slash.

XRS also allows the plug-in to create, delete and move folders. The commands for these functions are PM_CREATEFOLDER, PM_DELETEFOLDER and PM_MOVEFOLDER respectively.

To retrieve the folder tree (usually for display and user-navigation purposes), the PM_GETNEXTFOLDER command has to be used to obtain the next folder in the same level. To get the first sub-folder in the specified folder, use the PM_GETSUBFOLDER command.

When the active folder changes, the application will issue a PN_MEMFOLDER notification containing the absolute path to the new active folder.

DSP Control

Analog to Digital Conversion

To be able to perform analog to digital conversion (ADC), the RADIODSP_ADC flag must be set in the dwDspFeatures field of the DSPCAPS structure which can be accessed from the RADIODEVCAPS structure. There can be two versions, one for the receiver's DSP and one for the transmitter's DSP if either or both are supported. ADC from the receiver digitises data from the receiver's IF or demodulator output while on the transmitter it digitises data from the transmitter's input.

The capabilities of the ADC are defined by the existence of the RADIODSP_xBIT and the RADIODSP_xKHZ flags. For each flag specifies the number of bits and sampling rate is supported. The

ADC can possibly also support single and/or two channel (stereo) ADC, specified by the RADIODSP_MONO and RADIODSP_STEREO flags.

To start ADC, a PMx_DSPADCSTART command has to be issued to the application. The dwParam parameter has to include three RADIODSP_xxx flags that specify the sampling rate, bits per sample and number of channels.

Shortly after this command is executed (and assuming a success return value), PNx_DSPINBUFFULL notifications will be sent to the plug-in containing packets of digitised audio data in PCM format. The plug-in must use the data or make a copy of the data before returning from the notification (the data is freed after the plug-in returns from the notification).

To stop ADC, issue a PMx_DSPCLOSE command passing the return value from the PMx_DSPADCSTART command.

Digital to Analog Conversion

Digital to analog conversion (DAC) support is defined by the presence of the RADIODSP_DAC flag in the dwDspFeatures field of the DSPCAPS structure for both receivers and transmitters. On receivers, the DAC data is sent to the audio output while on transmitters, the data is sent to the transmitter's modulator input.

The capabilities of the DAC are defined by the same bits as the ADC (it is assumed that if the device supports both ADC and DAC, the capabilities are same or one is downgraded to the capabilities of the other).

To start DAC, a PMx_DSPDACSTART command as to be sent to the application. The dwParam parameter has to include three RADIODSP_xxx flags that specify the sampling rate, bits per sample and number of channels that the plug-in will provide data for.

Immediately after a successful return from the command, the plug-in can start sending digitised audio packets to the DAC using the PMx_DSPSENDBUF command. The command will return a 'Buffer ID' and any memory allocated for the command can be reused or freed. When the packet has been completely sent to the DAC, a PNx_DSPSENDBUFDONE notification is sent to the plug-in to let it know that the buffer has been sent.

When DAC has finished, issue a PMx_DSPCLOSE command passing the return value from the PMx_DSPDACSTART command.

DSP Programming

For devices that have programmable DSPs (not including fixed program DSPs), plug-ins can create DSP programs that can be uploaded to the DSP. Typical applications include filtering and decoding/encoding (depending on whether it is in a receiver or transmitter).

Support for programmable DSPs is defined by the presence of the RADIODSP_DSP flag in the dwDspFeatures field of the DSPCAPS structure. DSPs are always located between a ADC and a DAC where digitised audio data (from the ADC) is fed into the DSP, the DSP processes the data and sends it to the DAC. On receivers, the DSP processes audio data between the demodulator and the audio output. On transmitters, the DSP processes audio data between the input and the modulator. Depending on the DSP program, one end of the ADC-DSP-DAC chain may not be used such as in encoding or decoding digital data.

To upload and start a custom DSP program, the plug-in must first check that it supports the DSP in the device (the code is DSP dependant) by checking the szDspManufacturer and szDspProduct fields in the DSPCAPS structure. If the plug-in supports the device's DSP, the plug-in issues a PMx_DSPSTART command passing the DSP code in the lpData parameter.

DSP programs can receive PNx_DSPREQxxx notifications from the DSP itself, they can send data using the PMx_DSPSENDBUF and/or PMx_DSPSENDBYTE commands (the xxxBUF commands are generally faster than the xxxBYTE command) and receive data using the PMx_DSPADDINBUF and/or PMx_DSPREADBYTE commands.

On some receivers, the DSP supports processing from the IF through to the audio output. If the DSP supports programming from the IF input, the RADIODSP_IF flag is set in the dwDspFeatures field. Likewise with all DSP support, the code is device dependent and must be written specifically for the radio device and DSP.

Also, some DSPs support running of several programs simultaneously. This allows multiple calls to `PMx_DSPSTART` to upload and run several programs. Support for this feature is defined by the presence of the `RADIODSP_MULTIPLE` flag in the `dwDspFeatures` field. Each call to `PMx_DSPSTART` will return a unique handle that has to be used appropriately in all DSP commands.

XRS API Reference

Plug-in Functions

xrsPluginInit

Called when the application is started. Use this function to allocate the memory and resources shared by all instances of your plug-in.

C/C++:

```
int xrsPluginInit(int iXRSVer, PCHAR lpServerId, PCHAR lpName,
   int cbName);
```

Delphi:

```
function xrsPluginInit(iXRSVer: Integer; lpServerId, lpName: PChar;
   cbName: Integer): Integer;
```

Parameters

iXRSVer

Informs the plug-in the latest version of XRS the application supports. Any plug-ins that require an XRS version later than supplied should return a failure (zero) or adapt so only XRS specs supported are used.

lpServerId

Informs the plug-in of the server ID. The plug-in must validate the ID using the *xrsValidateServer* function.

lpName

Points to a buffer to accept the name of the plug-in. A plug-in can supply an empty string (NULL terminator only) to make the plug-in completely invisible (ie. not shown in any menu, list or from the PM_GETNEXTPLUGIN command).

cbName

Specifies the size of the buffer supplied at *lpName*. A plug-in must not put more than *cbName* bytes in the *lpName* buffer (including the NULL terminator).

Return Value

A plug-in can return 0 for failure (ie. do not load or use) or one of the following combinations that define the level of control the plug-in enjoys:

Bits 0 & 1:

0 = Failure

1 = Doesn't take control, but can occasionally change various settings.

2 = Takes temporary control, it should disable controls that is overriding.

3 = Takes full control, should hides/minimises/disables panel or appropriate controls.

Bits 2 & 3:

0 = Doesn't use DSP.

4 = Occasionally uses the DSP, it will have to share with other similar plug-ins.

8 = DSP parallel control, can operate with other plug-ins that also use the DSP.

12 = DSP full control, no other DSP based plug-ins can operate at the same time.

Note: *'DSP' represents the DSP facility available on the radio device itself - not a sound card device of the host computer.*

Bit 4:

Uses a sound device, cannot operate with other similar plug-ins.

Note: *The 'sound device' is typically a sound card facility of a personal computer.*

Bit 5:

Informs the application to start the plug-in immediately. (An 'auto-start' plug-in.)

Bit 6:

Informs the application not to show the plug-in in the plug-in list and/or menu but can still be accessed from the `PM_GETNEXTPLUGIN` command. (A 'hidden' plug-in.)

Bit 7:

Indicates that the plugin window handle will be returned in xrsPluginStart function and allows the XRS server to manipulate it (for example to embed to the application panel).

The plug-in should check the version information in the `iXRSVer` parameter to verify that it is compatible with the API capabilities provided by the application. The low-byte specifies the minor version number, and next most significant specifies the major version number. For example, for version 1.0 `iXRSVer` will equal 0x0100.

Next the plug-in ***must*** validate the server ID by calling `xrsValidateServer` with the `lpServerId` parameter and if it is valid, the function returns TRUE. If `xrsValidateServer` returns false, the plug-in must return zero.

No XRS API calls can take place in either direction until the initialisation completes successfully.

If a plug-in cannot initialise (resource allocation error, unsupported XRS version, or invalid XRS server), it must return zero.

xrsPluginDone

Called when the application is shutting down but before the plug-in is unloaded. It gives the plug-in a chance to cancel any outstanding I/O requests, delete threads it created, free memory and perform any other closing tasks.

C/C++:

```
void xrsPluginDone(void);
```

Delphi:

```
procedure xrsPluginDone;
```

xrsPluginStart

This is called when the user starts the plug-in from the menu or by a self-starting plug-in (see `xrsPluginInit`).

C/C++:

```
XRSRESULT xrsPluginStart(HWND hAppWnd, LPRADIODEVCAPS lpRadioDevCaps,
  PLUGINPROC lpPluginProc, PDWORD lpFilterFlags);
```

Delphi:

```
function xrsPluginStart(hAppWnd: HWND; lpRadioDevCaps: PRadioDevCaps;
  lpPluginProc: TPluginProc; var lpFilterFlags: Longint): TXrsResult;
```

Parameters

hAppWnd

Specifies the handle of the window that is starting the plug-in. This can be used to identify the device where multiple devices can start the plug-in.

lpRadioInfo

Points to a `RADIODEVCAPS` structure that contains the details of the device starting the plug-in.

lpPluginProc

Points to a caller-defined callback function for the plug-in to call, which will issue commands to the device. A full list of callback commands is defined later under '`PluginProc`'.

lpFilterFlags

Points to a DWORD that the plug-in may change to inform the application of the notifications it does not wish to receive. By specifying ignored notifications, the performance of the plug-in and application can be increased. If this value is not changed (or is set to zero), the plug-in will receive all notifications (and can choose to ignore any notifications it does not need to know about).

The `PN_CLOSE`, `PNR_SCANFINISHED` and several `PNR/T_DSPxxx` notifications cannot be filtered out.

The filter flags can be changed during a plug-in's operation by the `PM_FILTERFLAGS` command.

The filter values defined include:

Filter	Description
PNF_ALL	Do not receive any notifications (except unmaskable ones)
PNF_NONE	Receive all notifications
PNF_DISABLED	No PN_DISABLED notifications
PNF_POWER	No PN_POWER notifications
PNF_MEMORY	No PN_MEMxxx notifications
PNF_PLUGIN	No PN_PLUGINxxx notifications
PNF_ALLRX	No PNR_xxx notifications
PNF_RXFREQ	No PNR_FREQUENCY notifications
PNF_RXMODE	No PNR_MODE or PNR_MODEXDATA notifications
PNF_RXEXTOSC	No PNR_EXTOSC notifications
PNF_SLEVEL	No PNR_SLEVEL, PNR_SQUELCH or PNR_SQUELCHED notifications
PNF_RF	No PNR_ATTEN, PNR_PREAMP or PNR_RFINPUT notifications
PNF_IF	No PNR_IFSHIFT, PNR_AGC, PNR_IFGAIN, etc. notifications
PNF_RXAUDIO	No PNR_DEMODSIGNAL, PNR_VOLUME, PNR_MUTE, PNR_BALANCE, etc. notifications
PNF_SCANNER	No PNR_SCANNER notifications
PNF_RXDSP	No PNR_DSPxxx notifications
PNF_ALLTX	No PNT_xxx notifications
PNF_TXFREQ	No PNT_FREQUENCY notifications
PNF_TXMODE	No PNT_MODE or PNT_MODSRC notifications
PNF_TXEXTOSC	No PNT_EXTOSC notifications
PNF_TXAUDIO	No PNT_AUDIOPROC, PNT_AUDIOFILTER or PNT_ANTIVOX notifications
PNF_TXSETTINGS	No PNT_RFPOWER, PNT_SELCALL, PNT_XMTCTL, etc. notifications
PNF_MEASUREMENT	No PNT_MEASUREMENT notifications
PNF_TXDSP	No PNT_DSPxxx notifications

Return Value

If the plug-in starts successfully, the return value must be a process unique value defined by the plug-in (other than the failure code). Typically, the return value is a pointer to a unique memory location for each instance of a device. The plug-in can use *hAppWnd* as a device unique value.

If the plug-in fails, the plug-in must return -1 (or INVALID_HANDLE_VALUE in Win32).

Remarks

The caller can only call `xrsPluginStart` once for each instance, and after a `PM_CLOSED` callback command is received for the plug-in instance.

xrsPluginNotify

This is called every time a setting is changed in the associated device.

C/C++:

```
void xrsPluginNotify(HWND hAppWnd, int uMsg, DWORD dwData,
  int cbData, LPVOID lpData);
```

Delphi:

```
procedure xrsPluginNotify(hAppWnd: HWND; uMsg: Integer; dwData: Longint;
  cbData: Integer; lpData: Pointer);
```

Parameters

hAppWnd

Specifies the handle of the window of the device instance that is notifying the plug-in. This is the same as *hAppWnd* used in the `xrsPluginStart` function.

uMsg

The ID of the notification message (see Remarks below for all supported messages).

dwData

A 32-bit value associated with *uMsg*.

cbData

Size of the buffer pointed to *lpData* that is associated with *uMsg*.

lpData

A pointer to a buffer (that should not be modified) associated with *uMsg*. This can be NULL if this is not used.

Remarks

The notification messages that can be issued to a plug-in (from an application) include:

uMsg	Value (hex)	dwData	lpData	Page No.
PN_DISABLED	0000	Global disabled state of application interface	NULL	92
PN_POWER	0100	Power state (0 = off, 1 = on)	NULL	95
PN_MEMRECALL	0200	Memory number (0 – *dwMaxMemories*)	NULL	94
PN_MEMCHANGE	0201	Memory number	Pointer to MEMORYENTRY	93
PN_MEMFILE	0202	Not used	Pointer to file name	93
PN_MEMBANK	0203	Bank number	NULL	93
PN_MEMFOLDER	0204	Not used	Pointer to folder path	93
PN_PLUGINSTARTED	0300	Plug-in type (returned from **xrsInit**)	Pointer to name of plug-in	94
PN_PLUGINSTOPPED	0301	Plug-in type	Pointer to name of plug-in	95
Receiver Notifications				
PNR_FREQUENCY	0800	Receiver frequency in Hz	NULL	99
PNR_TRUNKFREQ	0801	Trunking system control frequency in Hz	NULL	108

PNR_TRACKID	0802	Radio ID to track in a trunking system.	NULL	108
PNR_TRUNKID	0803	Decoded radio ID from control frequency.	NULL	109
PNR_MODE	0900	RADIOMODE_xxx	NULL	103
PNR_MODEXDATA	0901	Depends on the mode	NULL	104
PNR_EXTOSC	0A00	Zero = internal, non-zero = external	NULL	98
PNR_SLEVEL	0B00	Signal level	NULL	107
PNR_SQUELCH	0B01	Enabled squelch settings (RXSQUELCH_xxx flags)	Pointer to SQUELCHSETTINGS	107
PNR_SQUELCHED	0B02	Zero = mute off, non-zero = mute on	NULL	108
PNR_RFINPUT	0C00	RF input number (1 to *iNumRfInputs*)	NULL	106
PNR_ATTEN	0C01	Atten level (0 to *iMaxAtten*)	NULL	100
PNR_PREAMP	0C02	Preamp level (0 to *iMaxPreamp*)	NULL	106
PNR_BANDWIDTH	0D00	IF bandwidth in Hz (*iMinIfBw* to *iMaxIfBw*)	NULL	101
PNR_AGC	0D01	AGC settings	NULL	99
PNR_IFGAIN	0D02	IF gain level (*iMinIfGain* to *iMaxIfGain*)	NULL	102
PNR_IFSHIFT	0D03	IF shift in Hz (up to +/- *iMaxIfShift*)	NULL	102
PNR_AFC	0D04	Zero = off, non-zero = on	NULL	99
PNR_VOLUME	0E00	Volume level (0 to *iMaxVolume*)	NULL	108
PNR_MUTE	0E01	Zero = off, non-zero = on	NULL	105
PNR_BALANCE	0E02	Audio balance (to +/- *iBalanceRange*)	NULL	101
PNR_MONO	0E03	0 = mono, 1 = stereo, -1 = forced mono	NULL	104
PNR_LOUD	0E04	Zero = off, non-zero = on	NULL	103
PNR_AUDIOSRC	0E05	RXAUDIOSRC_xxx	NULL	100
PNR_AUDIOFILTER	0E06	Filter type	Pointer to Filter settings	95
PNR_NOISEBLANKER	0E10	0 = off, -1 = auto, + = threshold	NULL	105
PNR_NOTCH	0E11	0 = off, -1 = auto, + = frequency in Hz	NULL	106
PNR_NOISEREDUCT	0E12	0 = off, + = type (*iMaxNoiseReduction)*	NULL	105
PNR_SCANNER	0F00	0 = stopped, 1 = scanning, 2 = paused	NULL	107
PNR_DSP	1000	0 = off, 1 = ADC, 2 = DAC, 3 = other	NULL	96
PNR_DSPINPUT	1001	Input number (0 – *iNumRxDspInputs*-1)	NULL	97
Transmitter Notifications				
PNT_FREQUENCY	1400	Transmitter frequency in Hz	NULL	99
PNT_MODE	1500	RADIOMODE_xxx	NULL	111
PNT_MODSRC	1501	TXMODSRC_xxx	NULL	112
PNT_EXTOSC	1600	Zero = internal, non-zero = external	NULL	98
PNT_AUDIOFILTER	1700	Filter number (0 = none)	Pointer to filter settings	95
PNT_AUDIOPROC	1701	Input processing type	Pointer to TXAUDIOPROC	110
PNT_ANTIVOX	1702	Anti-vox gain	NULL	109
PNT_TX	1800	Zero = not Txing, non-zero = is Txing	NULL	114
PNT_RFPOWER	1801	Transmitter power (0 to *iMaxTxPower*)	NULL	112
PNT_SELCALL	1802	Selective calling	Pointer to settings	113
PNT_XMTCTL	1803	Transmitter initiation control	NULL	114
PNT_MEASUREMENT	1900	Measurement type	Pointer to reading	110
PNT_DSP	1A00	0 = off, 1 = ADC, 2 = DAC, 3 = other	NULL	96
PNT_DSPINPUT	1A01	Input number (0 – *iNumTxDspInputs*-1)	NULL	97

The following messages cannot be filtered				
PN_CLOSE	4000	Not used	NULL	92
PN_MINIMIZED	4001	0 = normal, 1 = minimised	NULL	94
PN_VISIBLE	4002	0 = invisible (hidden), 1 = visible	NULL	95
PNR_SCANFINISHED	4010	Index of last frequency	Pointer to signal levels	107
PNR_FMWDATA	4011	Not used	Pointer to decoded data	101
PNR_DSPINBUFFULL	4020	Buffer ID	Pointer to buffer	97
PNR_DSPSENDBUFDONE	4021	Buffer ID	NULL	98
PNR_DSPREQUEST	4022	Request code (app defined)	NULL	98
PNR_DSPREQREAD	4023	Amount of data requested	NULL	97
PNR_DSPREQSEND	4024	Amount of data requested	NULL	98
PNT_DSPINBUFFULL	4030	Buffer ID	Pointer to buffer	97
PNT_DSPSENDBUFDONE	4031	Buffer ID	NULL	98
PNT_DSPREQUEST	4032	Request code (app defined)	NULL	98
PNT_DSPREQREAD	4033	Amount of data requested	NULL	97
PNT_DSPREQSEND	4034	Amount of data requested	NULL	98

XRS Functions

xrsCopyRadioDevCaps

This function allocates memory and copies the supplied `RADIODEVCAPS` structure into the allocated memory for a local copy of the radio device capabilities. This is typically used in the `xrsPluginStart` function.

C/C++:

```
LPRADIODEVCAPS xrsCopyRadioDevCaps(LPRADIODEVCAPS DevCaps);
```

Delphi:

```
function xrsCopyRadioDevCaps(DevCaps: PRadioDevCaps): PRadioDevCaps;
```

Parameters

DevCaps

Pointer to a `RADIODEVCAPS` structure, supplied in the `xrsPluginStart` function.

Return Value

Points to an allocated copy of the supplied `RADIODEVCAPS` structure. If memory could not be allocated, it returns NULL.

xrsFreeRadioDevCaps

This function frees memory allocated by `xrsCopyRadioDevCaps`.

C/C++:

```
void xrsFreeRadioDevCaps(LPRADIODEVCAPS DevCaps);
```

Delphi:

```
procedure xrsFreeRadioDevCaps(DevCaps: PRadioDevCaps);
```

Parameters

DevCaps

Points to a `RADIODEVCAPS` structure that was returned from `xrsCopyRadioDevCaps`.

xrsValidateServer

This function is provided for the plug-in to validate the server.

C/C++:

```
BOOL xrsValidateServer(PCHAR lpServerId);
```

Delphi:

```
function xrsValidateServer(lpServerId: PChar): Bool;
```

Parameters

lpServerId

The application's server ID passed in the `xrsPluginInit` call.

It is a text string in the format:

`XRS-XkQSCGjQ-`*`cccccc-nnnn`*

where *nnnn* represents the server's OEM ID.

Return Value

Non-zero if the server is valid, zero if the server is not.

PluginProc

This is an application-defined callback procedure for plug-ins to control various aspects of the application.

C/C++:

```
typedef DWORD (CALLBACK* PLUGINPROC)(DWORD, int, DWORD, int, LPVOID);
```

Delphi:

```
type
  TPluginProc = function (hPlugin: Longint; uMsg: Integer;
    dwParam: Longint; cbData: Integer; lpData: Pointer): Longint;
```

Parameters

hPlugin

A unique plug-in instance handle returned by `xrsPluginStart` so the application knows which plug-in is issuing the callback.

uMsg

The ID of the command for the application to process. These are defined under 'Remarks'.

dwParam

A 32-bit value associated with the *uMsg* command.

cbData

Size of the buffer pointed to *lpData* that is associated with *uMsg*.

lpData

A pointer to a buffer associated with *uMsg*. This can be NULL if not used. The buffer must have read and write access (regardless of whether the plug-in is expecting to use the data or not). The only exception is `PM_CLOSED`.

Return Value

If not successful, 0x80000000 (`PLUGIN_CB_FAIL`) is returned, otherwise the value depends on the command issued (see the table following).

Remarks

A full description for the commands that can be issued are described later in the section called 'Commands'.

A summary of the commands which can be issued from a plug-in to the calling application include:

uMsg	dwData	lpData	Success return value	Page no.
PM_CLOSED	Not used	Name of plug-in	0	57
PM_DISABLE	PD_xxx	NULL	0	58
PM_GETSETTINGS	PN_xxx	Depends on notification	Setting	62
PM_FILTERFLAGS	PNF_xxx	NULL	0	59
PM_MINIMIZE	Zero = restore, non-zero = minimise	NULL	0	
PM_VISIBLE	Zero = hide, non-zero = show	NULL	0	
PM_POWER	Zero = off, non-zero = on	NULL	0	64
Receiver Commands				
PMR_FREQUENCY	The receiver's frequency in Hz (changes the display frequency)	NULL	0	73
PMR_FREQ	The receiver's frequency in Hz (does not change the display frequency)	NULL	0	72
PMR_TRUNKFREQ	The trunking control frequency in Hz	NULL	0	85
PMR_TRACKID	The radio ID to track	NULL	0	84
PMR_EXTOSC	Zero = internal, Non-zero = external	NULL	0	72
PMR_MODE	RADIOMODE_xxx	NULL	0	79
PMR_MODEXDATA	Depends on mode	NULL	0	80
PMR_SQUELCH	RXSQUELCH_xxx	Ptr to SQUELCHSETTINGS	0	83
PMR_RFINPUT	RF input number (1 to *iNumRfInputs*)	NULL	0	83
PMR_PREAMP	RF pre-amplification level (0 to *iMaxPreamp*)	NULL	0	82
PMR_ATTEN	RF attenuation level (0 to *iMaxAtten*)	NULL	0	74
PMR_BANDWIDTH	IF bandwidth in Hz	NULL	0	76
PMR_AGC	AGC settings	NULL	0	73
PMR_IFGAIN	IF gain level (*iMaxIfGain* to *iMaxIfGain*)	NULL	0	77
PMR_IFSHIFT	IF shift in current mode (up to +/- *iMaxIfShift*)	Ptr to RADIOMODE_xxx if to apply to other mode	0	78
PMR_AFC	Zero = off, non-zero = on	NULL	0	73
PMR_VOLUME	Volume level (0 to *iMaxVolume*)	NULL	0	84
PMR_MUTE	Zero = off, non-zero = on	NULL	0	81
PMR_BALANCE	Left/right balance (+/-*iBalanceRange*)	NULL	0	75
PMR_MONO	Zero = stereo, non-zero = forced mono	NULL	0	79
PMR_LOUD	Zero = off, non-zero = on	NULL	0	78

PMR_AUDIOSRC	RXAUDIOSRC_xxx	NULL	0	75
PMR_AUDIOFILTER	Filter type	Filter settings	0	67
PMR_NOISEBLANKER	0 = off, -1 = auto, + = threshold	NULL	0	81
PMR_NOTCH	0 = off, -1 = auto, + = frequency in Hz	NULL	0	82
PMR_NOISEREDUCT	0 = off, + = type (1 to *iMaxNoiseReduction)*	NULL	0	81
PMR_BLOCKSCAN	High word = squelch (-1 = not used) Low word = scan rate (f/s)	Pointer to frequencies	0	76
PMR_STOPSCAN	Not used	Optional pointer to receive signal levels	Last index	84
PMR_DSPADCSTART	Rate, bits & chnls (RADIODSP_xxx)	NULL	DSP handle	68
PMR_DSPDACSTART	Rate, bits & chnls (RADIODSP_xxx)	NULL	DSP handle	69
PMR_DSPSTART	Not used	Pointer to DSP code	DSP handle	71
PMR_DSPCLOSE	DSP handle	NULL	0	69
PMR_DSPSENDBUF	DSP handle	Pointer to buffer to send	Buffer ID	71
PMR_DSPADDINBUF	DSP handle	NULL (*cbSize* = buffer size)	Buffer ID	69
PMR_DSPSENDBYTE	DSP handle	Pointer to data to send to DSP	Number of bytes sent	71
PMR_DSPREADBYTE	DSP handle	Pointer to buffer to receive data from DSP	Number of bytes sent	70
PMR_DSPINPUT	DSP input number (0 to *iNumRxDspInputs*)	NULL	0	70
Transmitter Commands				
PMT_FREQUENCY	The transmitter's frequency in Hz (changes the display frequency)	NULL	0	73
PMT_FREQ	The transmitter's frequency in Hz (does not change the display frequency)	NULL	0	72
PMT_EXTOSC	Zero = internal, Non-zero = external	NULL	0	72
PMT_MODE	RADIOMODE_xxx	Pointer to MODPARAMS	0	86
PMT_MODSRC	Audio input source (TXMODSRC_xxx)	NULL	0	88
PMT_AUDIOFILTER	Filter type	Pointer to filter settings	0	67
PMT_AUDIOPROC	Processing flags	Pointer to TXAUDIOPROC	0	86
PMT_ANTIVOX	Anti-vox gain (0 to *iMaxAntiVox*)	NULL	0	85
PMT_RFPOWER	Transmitter power (0 to *iMaxTxPower*)	NULL	0	88
PMT_SELCALL	Selective calling	Pointer to settings	0	89
PMT_TX	Zero = not transmitting, non-zero = transmitting	NULL	0	90
PMT_XMTCTL	Transmitter initiation & release time	NULL	0	90
PMT_DSPADCSTART	Rate, bits & chnls (RADIODSP_xxx)	NULL	0	68
PMT_DSPDACSTART	Rate, bits & chnls (RADIODSP_xxx)	NULL	0	69
PMT_DSPSTART	Not used	Pointer to DSP code	0	71
PMT_DSPCLOSE	Not used	NULL	0	69
PMT_DSPSENDBUF	Not used	Pointer to buffer to send	Buffer ID	71
PMT_DSPADDINBUF	Not used	NULL (*cbSize* = buffer size)	Buffer ID	69
PMT_DSPSENDBYTE	Byte to send if *lpData* not NULL	Optional pointer to data to send to DSP	Number of bytes sent	71
PMT_DSPREADBYTE	Not used	Optional pointer to buffer to receive data from DSP	Number of bytes read	70
PMT_DSPINPUT	DSP input number (0 to *iNumTxDspInputs*)	NULL	0	70
Memory Commands				

PM_RECALLMEM	Memory no. (0 to *dwMaxMemories*)	NULL	0	64
PM_STOREMEM	Memory number	Ptr to MEMORYENTRY	0	66
PM_GETMEM	Memory number	Ptr to MEMORYENTRY	0	60
PM_GETNUMMEMS	Not used	NULL	No. of memory records	62
PM_GETNEXTMEM	Memory number (-1 = first)	NULL	Next mem no.	61
PM_GETMEMFILE	Not used	Pointer to buffer to receive file name	0	60
PM_SETMEMFILE	Not used	Pointer to file name	0	65
PM_SELECTBANK	Bank number (0 to *iNumBanks*)	NULL	0	65
PM_OPENFOLDER	Not used	Pointer to folder path	0	64
PM_GETNEXTFOLDER	Not used	Pointer to folder path	Length of path	60
PM_GETSUBFOLDER	Not used	Pointer to folder path	Length of path	62
PM_CREATEFOLDER	Not used	Pointer to new folder path	0	57
PM_DELETEFOLDER	Not used	Pointer to folder path	0	58
PM_MOVEFOLDER	Not used	New destination path	0	63
Plug-in Commands				
PM_GETNEXTPLUGIN	Not used	Pointer to buffer for name	Plug-in type	61
PM_STARTPLUGIN	Not used	Pointer to plug-in name	0: started, 1: already started	66
PM_STOPPLUGIN	Not used	Pointer to plug-in name	0: stopped, 1: not running	66

XRS Structures

AGCEXCAPS

The AGCEXCAPS structure is used when the receiver supports the setting extended AGC parameters. Support for extended AGC parameters is specified by the existence of the RADIOTXCAPS_ADJAGC flag in the *dwRxFeatures* field of the RADIODEVCAPS structure. The *lpAgcExCaps* field refers to this structure (if it is not NULL).

The AGCEXCAPS structure specifies the minimum and maximum values allowable for each part of the AGC parameters.

C/C++:

```
typedef struct _AGCEXCAPS {
   int    iMinAgcAttack;
   int    iMaxAgcAttack;
   int    iMinAgcHold;
   int    iMaxAgcHold;
   int    iMinAgcDecay;
   int    iMaxAgcDecay;
} AGCEXCAPS, FAR *LPAGCEXCAPS;
```

Delphi:

```
type
   PAgcExCaps = ^TAgcExCaps;
   TAgcExCaps = packed record
       iMinAgcAttack: Integer;
       iMaxAgcAttack: Integer;
       iMinAgcHold: Integer;
       iMaxAgcHold: Integer;
       iMinAgcDecay: Integer;
       iMaxAgcDecay: Integer;
   end;
```

Fields

iMinAgcAttack

Specifies the minimum AGC attack time in 1ms intervals.

iMaxAgcAttack

Specifies the maximum AGC attack time. If this (and *iMinAgcAttack*) is zero, then the receiver does not support adjustable AGC attack times.

iMinAgcHold

Specifies the minimum AGC hold time in 1ms intervals.

iMaxAgcHold

Specifies the maximum AGC hold time. If this (and *iMinAgcHold*) is zero, then the receiver does not support adjustable AGC hold times.

iMinAgcDecay

Specifies the minimum AGC decay time in 1ms intervals.

iMaxAgcDecay

Specifies the maximum AGC decay time. If this (and *iMinAgcDecay*) is zero, then the receiver does not support adjustable AGC decay times.

AGCEXPARAMS

The AGCEXPARAMS structure is used to specify extended AGC parameters in the PMR_AGC command and PNR_AGC notification.

C/C++:

```
typedef struct _AGCEXPARAMS {
   DWORD  dwAgcAttack;
   DWORD  dwAgcHold;
   DWORD  dwAgcDecay;
} AGCEXPARAMS, FAR *LPAGCEXPARAMS;
```

Delphi:

```
type
   PAgcExParams = ^TAgcExParams;
   TAgcExParams = record
       dwAgcAttack: Longint;
       dwAgcHold: Longint;
       dwAgcDecay: Longint;
   end;
```

Fields

dwAgcAttack

Specifies the AGC attack time in 1ms intervals.

dwAgcHold

Specifies the AGC hold time in 1ms intervals.

dwAgcDecay

Specifies the AGC decay time in 1ms intervals.

DEMODDEF

The DEMODDEF structure describes a supported demodulation mode and its associated attributes. This is included as part of the RADIODEVCAPS structure passed in the xrsPluginStart function.

C/C++:

```
typedef struct _DEMODDEF {
   int    iMode;
   int    iMaxScanRate;
   DWORD  dwMinIfBw;
   DWORD  dwMaxIfBw;
   int    iIfBwStep;
   DWORD  dwMaxIfShift;
   DWORD  dwMaxExData;
} DEMODDEF, FAR *LPDEMODDEF;
```

Delphi:

```
type
   PDemodDef = ^TDemodDef;
   TDemodDef = record
       iMode: Integer;
       iMaxScanRate: Integer;
       dwMinIfBw: Longint;
       dwMaxIfBw: Longint;
       iIfBwStep: Integer;
       dwMaxIfShift: Longint;
       dwMaxExData: Longint;
   end;
```

Fields

iMode

Specifies the mode (RADIOMODE_xxx). A mode can be specified more than once if each mode has a different fixed IF bandwidth (where *dwMinIfBw* is –1). This value is used in the PMR_MODE command.

iMaxScanRate

Specifies the maximum scanning rate for this mode.

dwMinIfBw

Specifies the minimum IF bandwidth that can be set in this mode in Hz. If it is –1, the bandwidth is fixed and cannot be adjusted with the PMR_BANDWIDTH command.

dwMaxIfBw

If *dwMinIfBw* is positive (or zero), this specifies the maximum IF bandwidth for this mode in Hz. If *dwMinIfBw* is –1, this specifies the IF bandwidth for this mode.

iIfBwStep

If the IF bandwidth is adjustable for the mode, this specifies the bandwidth adjustment granularity in Hz.

dwMaxIfShift

If the mode supports adjustable IF shift, this value specifies the maximum shift range (+ or -) from the centre in Hz. If the mode does not support IF shift, this is set to zero.

dwMaxExData

Most modes have an extended attribute that can be set. This parameter specifies the maximum value that can be set in the PMR_MODEXDATA command.

In CW, the extended data controls the BFO offset. This field specifies the maximum BFO range, and is zero if it is not supported.

In FM modes, the extended data controls with audio base-band width in Hz. This field specifies the maximum base-band width or is zero if it is not adjustable.

For all other modes, this field is reserved.

DEMODSIGNALDATA

The DEMODSIGNALDATA structure is used to pass the samples from a digital demodulator point to a plug-in through the PNR_DEMODSIGNAL message. The XRS server receives the samples from a demodulator plug-in using the same structure and dispatches it to all other plug-ins.

C/C++:

```
typedef struct _DEMODSIGNALDATA {
     int    iSamplingRate;
     int    iBitsPerSample;
     int    iNumChannels;
     int    iNumSamplesSets;
     BYTE   Samples[1];
  } DEMODSIGNALDATA, FAR *LPDEMODSIGNALDATA;
```

Delphi:

```
type
  PDemodSignalData = ^TDemodSignalData;
  TDemodSignalData = record
     iSamplingRate: Integer;
     iBitsPerSample: Integer;
     iNumChannels: Integer;
     iNumSamplesSets: Integer;
     Samples: array [0..0] of Char;
  end;
```

Fields:

iSamplingRate

Specifies the sampling rate corresponding to the samples in the structure.

iBitsPerSample

Specifies the size of each sample stored in the structure in bits. It must be a multiple of 8.

iNumChannels

Specifies the number of channels for which the samples are interlaced in the structure.

iNumSamplesSets

Specifies the number of sets of samples contained in the structure. Such a set contains one sample for each channel.

Samples

The actual samples contained in the structure. The total size of this field is given by:

iNumSamplesSets * iNumChannels * iBitsPerSample / 8

DSPCAPS

The DSPCAPS structure is used to specify the DSP used in the receiver and/or transmitter and its capabilities in the device. The receiver's DSP capabilities are referred from the *lpRxDspCaps* field and the transmitter's from the *lpTxDspCaps* field of the RADIODEVCAPS structure.

C/C++:

```
typedef struct _DSPCAPS {
   CHAR   szDspManufacturer[32];
   CHAR   szDspProduct[32];
   DWORD  dwDspFeatures;
   int    iNumDspInputs;
   int    iCodeWordSize;
   int    iDataWordSize;
   int    iExtWordSize;
   DWORD  dwCodeSize;
   DWORD  dwDataSize;
   DWORD  dwExtSize;
} DSPCAPS, FAR *LPDSPCAPS;
```

Delphi:

```
type
   PDspCaps = ^TDspCaps;
   TDspCaps = record
       szDspManufacturer: array [0..31] of Char;
       szDspProduct: array [0..31] of Char;
       dwDspFeatures: Longint;
       iNumDspInputs: Integer;
       iCodeWordSize: Integer;
       iDataWordSize: Integer;
       iExtWordSize: Integer;
       dwCodeSize: Longint;
       dwDataSize: Longint;
       dwExtSize: Longint;
   end;
```

Fields

szDspManufacturer

Specifies the name of the manufacturer who made the device's DSP (or ADC/DAC if a DSP doesn't exist). For ADCs and/or DACs, this field is optional (as they cannot be programmed).

szDspProduct

Specifies the product name of the device's DSP (or ADC/DAC if a DSP doesn't exist). For ADCs and/or DACs, this field is optional (as they cannot be programmed).

For plug-ins that provide DSP programs, this field must be checked as each product is generally unique in its hardware implementation.

dwDspFeatures

Specifies a range of flags that specify what the DSP, ADC and/or DACs support and which of these are supported:

`RADIODSP_ADC` ..... *supports analog to digital conversion (recording)*
`RADIODSP_DAC` ..... *supports digital to analog conversion (playback)*
`RADIODSP_DSP` ..... *supports DSP functionality (programmable)*
`RADIODSP_AUDIO` ..... *supports audio DSP functionality*
`RADIODSP_IF` ..... *supports IF DSP functionality*
`RADIODSP_MULTIPLE` ..... *supports multiple DSP operations*

The following only apply to digital recording and playback (`RADIODSP_ADC` and `RADIODSP_DAC`):

`RADIODSP_8BIT` ..... *supports 8 bit sampling*
`RADIODSP_16BIT` ..... *supports 16 bit sampling*
`RADIODSP_24BIT` ..... *supports 24 bit sampling*
`RADIODSP_32BIT` ..... *supports 32 bit sampling*

`RADIODSP_8KHZ` ..... *supports 8 kHz sampling rate*
`RADIODSP_11KHZ` ..... *supports 11.025 kHz*
`RADIODSP_16KHZ` ..... *supports 16 kHz*
`RADIODSP_22KHZ` ..... *supports 22.05 kHz*
`RADIODSP_32KHZ` ..... *supports 32 kHz*
`RADIODSP_44KHZ` ..... *supports 44.1 kHz*
`RADIODSP_48KHZ` ..... *supports 48 kHz*
`RADIODSP_64KHZ` ..... *supports 64 kHz*
`RADIODSP_96KHZ` ..... *supports 96 kHz*

`RADIODSP_MONO` ..... *supports single channel sampling*
`RADIODSP_STEREO` ..... *supports two channel sampling*

iNumDspInputs

Specifies the number of selectable inputs to the ADC and/or DSP.

iCodeWordSize

Specifies the number of bits per word in program memory on the DSP.

iDataWordSize

Specifies the number of bits per word in data memory on the DSP.

iExtWordSize

Specifies the number of bits per word in external memory used by the DSP.

dwCodeSize

Specifies how many words are available in program memory on the DSP.

dwDataSize

Specifies how many words are available in data memory on the DSP.

dwExtSize

Specifies how many words are available in external memory for the DSP.

FREQRANGE

The FREQRANGE structure describes a support frequency range (or band) and any associated attributes of the frequency for a radio device. This is included as part of the RADIODEVCAPS structure passed in the xrsPluginStart function.

C/C++:

```
typedef struct _FREQRANGE {
   DWORD  dwMinFreqkHz;
   DWORD  dwMaxFreqkHz;
   int    iRfInputs;
} FREQRANGE, FAR *LPFREQRANGE;
```

Delphi:

```
type
   PFreqRange = ^TFreqRange;
   TFreqRange = record
       dwMinFreqkHz: Longint;
       dwMaxFreqkHz: Longint;
       iRfInputs: Integer;
   end;
```

Fields

dwMinFreqkHz

Specifies the minimum tunable frequency for this range in kHz.

dwMaxFreqkHz

Specifies the maximum tunable frequency for this range in kHz.

iRfInputs

An array of bits (bit 0 = RF input #1) that specifies which RF input(s) can be used to receive signals within this range.

This field is not used for transmitter bands.

GRAPHEQCAPS

The GRAPHEQCAPS structure is used to specify the capabilities of a graphic equaliser if supported on a receiver and/or transmitter. It is accessed from the *lpGraphEqCaps* field of the RADIODEVCAPS structure.

C/C++:

```
typedef struct _GRAPHEQCAPS {
   int    iLevelRange;
   int    iLevelStep;
   int    iNumFreqs;
   int    iFreq[1];
} GRAPHEQCAPS, FAR *LPGRAPHEQCAPS;
```

Delphi:

```
type
   PGraphEqCaps = ^TGraphEqCaps;
   TGraphEqCaps = record
       iLevelRange: Integer;
       iLevelStep: Integer;
       iNumFreqs: Integer;
       iFreq: array [0..0] of Integer;
   end;
```

Fields

iLevelRange

Specifies the maximum adjustment range (boost and cut) of each supported frequency.

If the RADIOCAL_EQUALIZER flag is specified in the *dwCalibrated* field of the RADIODEVCAPS structure, then this value is a multiple of 0.1 dB.

iLevelStep

Specifies the granularity of the level adjustment.

iNumFreqs

Specifies the number of adjustment frequencies.

iFreq

Specifies an array of centre frequencies (in Hz) that can be adjusted.

MEMORYENTRY

The MEMORYENTRY structure is used for transferring memory information between an application and a plug-in.

C/C++:

```
typedef struct _TXSCHEDULE {
   DWORD dwDays;       // bit 0 = Sunday .. bit 6 = Saturday
   DWORD dwStartTime;         // in seconds from midnight
   DWORD dwStopTime;
} TXSCHEDULE, FAR *LPTXSCHEDULE;


typedef struct _MEMORYENTRY {
   DWORD  cbSize;
   CHAR   szName[64];

   DWORD  dwFrequency;
   DWORD  dwStepSize;
   DWORD  dwMode;
   DWORD  dwModeExData;
   DWORD  dwSquelch;

   DWORD  dwRfInput;
   DWORD  dwAtten;
   DWORD  dwPreamp;

   DWORD  dwBandwidth;
   DWORD  dwAgc;
   DWORD  dwIfGain;
   DWORD  dwIfShift;
   DWORD  dwAfc;

   DWORD  dwNumHits;
   DWORD  dwLastSLevel;
   DWORD  dwMaxSLevel;

   DWORD  dwNumSchedules;
   DWORD  dwScheduleOffset;

   DWORD  dwGroups;

   DATE   dtStored;
   DATE   dtModified;
   DATE   dtRecalled;

   DWORD  fLockout;

   CHAR   szCallsign[32];
   CHAR   szComments[256];
} MEMORYENTRY, FAR *LPMEMORYENTRY;
```

Delphi:

```
type
```

```
PTxSchedule = ^TTxSchedule;
TTxSchedule = record
    dwDays: Longint;
    dwStartTime: Longint;
    dwStopTime: Longint;
end;

PMemoryEntry = ^TMemoryEntry;
TMemoryEntry = record
    cbSize: Longint;
    szName: array [0..63] of Char;

    dwFrequency: Longint;
    dwStepSize: Longint;
    dwMode: Longint;
    dwModeExData: Longint;
    dwSquelch: Longint;

    dwRfInput: Longint;
    dwAtten: Longint;
    dwPreamp: Longint;

    dwBandwidth: Longint;
    dwAgc: Longint;
    dwIfGain: Longint;
    dwIfShift: Longint;
    dwAfc: Longint;

    dwNumHits: Longint;
    dwLastSLevel: Longint;
    dwMaxSLevel: Longint;

    dwNumSchedules: Longint;
    dwScheduleOffset: Longint;

    dwGroups: Longint;

    dtStored: TDateTime;
    dtModified: TDateTime;
    dtRecalled: TDateTime;

    fLockout: LongBool;

    szCallsign: array [0..31] of Char;
    szComments: array [0..255] of Char;
end;
```

Fields

cbSize

Specifies the size of the `MEMORYENTRY` structure in bytes (not including any information appended to the end such as transmission schedules).

szName

Specifies the name of the memory record that the user has nominated.

This field is only supported if the `RADIOMEM_NAME` flag is specified in the *dwMemFeatures* field in the `RADIODEVCAPS` structure.

dwFrequency

Specifies the frequency (in Hz) stored in the memory record. If bit 31 is set, the frequency in the low 31 bits are multiplied by ten (allowing up to a 21 GHz range).

This field must be supported and be greater than zero (unless deleting a memory record).

dwStepSize

Specifies the step size to set when the record is recalled in Hz. If the value is zero, the step size is not specified.

This field is only supported if the RADIOMEM_STEPSIZE flag is specified in the *dwMemFeatures* field in the RADIODEVCAPS structure.

dwMode

Specifies the mode to set when the record is recalled. The value corresponds to a RADIOMODE_xxx constant. If the value is less than zero, the mode is not specified.

This field is supported if the RADIOMEM_MODE flag is specified in the *dwMemFeatures* field.

dwModeExData

Specifies mode dependant data.

dwSquelch

Specifies the squelch level to set when the record is recalled. If the value is less than zero, the squelch is not specified.

This field is supported if the RADIOMEM_SQUELCH flag is specified in the *dwMemFeatures* field.

dwRfInput

Specifies which RF input to use when the record is recalled. If zero is specified, the RF input selection is not changed.

This field is supported if the RADIOMEM_RFINPUT flag is specified in the *dwMemFeatures* field.

dwAtten

Specifies the attenuator setting when the record is recalled. The range will correspond to the receiver's attenuator range (and RADIOCAL_ATTEN will be set if the value is in dB). If the value is less than zero, the attenuator level is not specified.

This field is supported if the RADIOMEM_ATTEN flag is specified in the *dwMemFeatures* field.

dwPreamp

Specifies the preamplifier gain level when the record is recalled. The range will correspond to the receiver's preamplifier range (and RADIOCAL_PREAMP will be set if the value is in dB). If the value is less than zero, the preamplifier level is not specified.

This field is supported if the RADIOMEM_PREAMP flag is specified in the *dwMemFeatures* field.

dwBandwidth

Specified the IF bandwidth (in Hz) to set when the record is recalled. If the value is zero, the bandwidth is not specified.

This field is supported if the RADIOMEM_BANDWIDTH flag is specified in the *dwMemFeatures* field.

dwAgc

Specifies the AGC settings to set when the record is recalled. The range and format supported is specified by the receiver's AGC capabilities. If the value is less than zero, the AGC settings are not specified.

The field is supported if the RADIOMEM_AGC flag is specified in the *dwMemFeatures* field.

dwIfGain

Specifies the IF gain level the if AGC is deactivated or the limits the maximum gain which can be achieved by AGC action. If the IF gain is not stored, the value is set to 0x80000000.

The field is supported if the RADIOMEM_IFGAIN flag is specified in the *dwMemFeatures* field.

dwIfShift

Specifies the amount of IF shift to apply (in Hz) when the record is recalled. If the IF shift is not stored, the value is set to 0x80000000.

The field is supported if the RADIOMEM_IFSHIFT flag is specified in the *dwMemFeatures* field.

fAfc

Specifies whether the AFC is active or not when the record is recalled. If zero is specified, the AFC is deactived, a positive value activates the AFC and a negative value does not change the AFC setting.

The field is supported if the RADIOMEM_AFC flag is specified in the *dwMemFeatures* field.

dwNumHits

Specifies the number of times the memory scanner has paused at this record due to the signal level being above the squelch level.

The field is supported if the RADIOMEM_HITCOUNT flag is specified in the *dwMemFeatures* field.

dwLastSLevel

Specifies the last recorded signal level for the associated frequency.

The field is supported if the RADIOMEM_SLEVEL flag is specified in the *dwMemFeatures* field.

dwMaxSLevel

Specifies the maximum recorded signal level for the associated frequency.

The field is supported if the RADIOMEM_SLEVEL flag is specified in the *dwMemFeatures* field.

dwNumSchedules

Specifies the number of transmission schedules stored in the record. An array of TXSCHEDULE entries (the number of entries specified by this field) follows the MEMORYENTRY structure, its location specified by the *dwScheduleOffset* field.

The field is supported if the RADIOMEM_SCHEDULE flag is specified in the *dwMemFeatures* field.

dwScheduleOffset

Specifies the offset from the beginning of the MEMORYENTRY structure to the TXSCHEDULE array.

The field is supported if the RADIOMEM_SCHEDULE flag is specified in the *dwMemFeatures* field.

dwGroups

Specifies the group(s) allocation for the record.

The field is supported if the RADIOMEM_GROUPS flag is specified in the *dwMemFeatures* field.

dtStored

Specifies the date and time the record was initially stored into the memory. If this is zero, the field is not supported.

The field is supported if the RADIOMEM_DATETIME flag is specified in the *dwMemFeatures* field.

dtModified

Specifies the date and time the record was last modified. If this is zero, the field is not supported.

The field is supported if the RADIOMEM_DATETIME flag is specified in the *dwMemFeatures* field.

dtRecalled

Specifies the date and time the record was last recalled. If this is zero, the field is not supported.

The field is supported if the RADIOMEM_DATETIME flag is specified in the *dwMemFeatures* field.

fLockout

Specifies whether the record is excluded from memory scans or not.

The field is supported if the RADIOMEM_LOCKOUT flag is specified in the *dwMemFeatures* field.

szCallsign

Specifies the callsign associated with the frequency stored in the record.

The field is supported if the `RADIOMEM_CALLSIGN` flag is specified in the *dwMemFeatures* field.

szComments

Specifies the comment the user has included in the record.

The field is supported if the `RADIOMEM_COMMENT` flag is specified in the *dwMemFeatures* field.

MODDEF

The `MODDEF` structure describes a supported modulation mode and its associated attributes. This is included as part of the `RADIODEVCAPS` structure passed in the `xrsPluginStart` function.

C/C++:

```
typedef struct _MODDEF {
   int    iMode;
   DWORD dwMaxParam1;
   DWORD dwMaxParam2;
   DWORD dwMaxParam3;
   DWORD dwMaxParam4;
} MODDEF, FAR *LPMODDEF;
```

Delphi:

```
type
   PModDef = ^TModDef;
   TModDef = record
      iMode: Integer;
      dwMaxParam1: Longint;
      dwMaxParam2: Longint;
      dwMaxParam3: Longint;
      dwMaxParam4: Longint;
   end;
```

Fields

iMode

Specifies the mode (`RADIOMODE_xxx`). The meaning of the remainder of the fields depends on this value.

dwMaxParam1

CW:

Not used.

LSB, USB:

Specifies the maximum 'peak envelope power' supported. This is zero if it is not adjustable.

AM:

Specifies the maximum 'modulation depth'. This is zero if it is not adjustable.

FMN, FMM, FMW:

Specifies the maximum frequency deviation that can be set. This is zero if it is not adjustable.

FSK:

Specifies the highest base frequency for an FSK transmission in Hz.

DAB:

Specifies the supported digital audio broadcasting standards. Each set bit represents supported standards:

0 = Eureka 147
1 = IBOC
2 = WordSpace
3 = DRM

dwMaxParam2

CW, LSB, USB, AM, DAB:

Not used.

FMN, FMM, FMW:

Specifies the maximum base frequency that can be set. This is zero if it is not adjustable.

FSK:

Specifies the maximum shift frequency in Hz.

dwMaxParam3

CW, LSB, USB, AM, FMN, FMW, DAB:

Not used.

FMW:

Specifies the maximum pilot tone frequency that can be set. This is zero if it is not supported.

FSK:

Specifies the maximum baud rate for the transmission.

dwMaxParam4

CW, LSB, USB, AM, FMN, FMM, FMW, DAB:

Not used.

FSK:

Specifies the number of 'shapes' supported.

MODPARAMS

The MODPARAMS structure is used to specify the general modulation parameters used in the PMT_MODE command (and receive in the PNT_MODE notification).

C/C++:

```
typedef struct _MODPARAMS {
   DWORD dwPrimaryModeParam1;
   DWORD dwPrimaryModeParam2;
   DWORD dwPrimaryModeParam3;
   DWORD dwPrimaryModeParam4;

   DWORD dwSecondaryCarrierFreq;

   DWORD dwSecondaryModeParam1;
   DWORD dwSecondaryModeParam2;
   DWORD dwSecondaryModeParam3;
   DWORD dwSecondaryModeParam4;
} MODPARAMS, FAR *LPMODPARAMS;
```

Delphi:

```
type
   PModParams = ^TModParams;
   TModParams = record
       dwPrimaryModeParam1: Longint;
       dwPrimaryModeParam2: Longint;
       dwPrimaryModeParam3: Longint;
       dwPrimaryModeParam4: Longint;

       dwSecondaryCarrierFreq: Longint;
```

```
        dwSecondaryModeParam1: Longint;
        dwSecondaryModeParam2: Longint;
        dwSecondaryModeParam3: Longint;
        dwSecondaryModeParam4: Longint;
    end;
```

Fields

The meaning for each of these parameters (one to four) depends on the mode.

dwPrimaryModeParam1

CW:

Not used.

LSB, USB:

Specifies the 'peak envelope power'. The *dwMaxParam1* field in the MODDEF structure specifies the maximum limit.

If the RADIOCAL_SSBMODPEP flag is set in the *dwCalibrated* field of the RADIODEVCAPS structure, this value is specified as a percentage of the max.

AM:

Specifies the 'modulation depth'. The maximum limit is specifies by the *dwMaxParam1* field in the MODDEF structure.

If the RADIOCAL_AMMODDEPTH flag is set in the *dwCalibrated* field of the RADIODEVCAPS structure, this value is specified as a percentage of the max.

FMN, FMM, FMW:

Specifies the maximum frequency deviation either side of the carrier.

If the RADIOCAL_FMDEV flag is set in the *dwCalibrated* field of the RADIODEVCAPS structure, this field is specified in Hz.

FSK:

Specifies the lower frequency of the FSK transmission in Hz.

DAB:

Specifies the Digital Audio Broadcasting standard:

0 = Eureka 147
1 = IBOC
2 = WordSpace
3 = DRM

dwPrimaryModeParam2

CW, LSB, USB, AM, DAB:

Not used.

FMN, FMM, FMW:

Specifies the base bandwidth of the input signal in Hz.

FSK:

Specifies the frequency shift from the lower frequency.

dwPrimaryModeParam3

CW, LSB, USB, AM, FMN, FMM, DAB:

Not used.

FMW:

Specifies whether the transmission is in stereo or not. If this is zero, the transmission is in mono (no pilot tone or 2nd channel sub-carrier is transmitted). If this is one, a pilot tone is transmitted at 19 kHz with the 2nd channel sub-carrier transmitted at 38 kHz.

If the transmitter supports variable pilot tone frequencies, this specifies the pilot tone frequency (the sub-carrier is double the pilot tone frequency).

The *dwMaxParam3* field of the MODDEF structure defines the maximum value. If the RADIOCAL_FMWPILOTTONE flag is specified in the *dwCalibrated* field, then this value is in Hz.

FSK:

Specifies the baud rate of the transmission.

dwPrimaryModeParam4

CW, LSB, USB, AM, FMN, FMM, FMW, DAB:

Not used.

FSK:

Specifies shaping of the frequency transitions.

dwSecondaryCarrierFreq

Specifies the frequency of the secondary (or sub) carrier.

dwSecondaryModeParam1

The same as *dwPrimaryModeParam1* but specifies the parameters for the secondary sub-carrier.

dwSecondaryModeParam2

The same as *dwPrimaryModeParam2* but specifies the parameters for the secondary sub-carrier.

dwSecondaryModeParam3

The same as *dwPrimaryModeParam3* but specifies the parameters for the secondary sub-carrier.

dwSecondaryModeParam4

The same as *dwPrimaryModeParam4* but specifies the parameters for the secondary sub-carrier.

PARAEQCAPS

The PARAEQCAPS structure is used to specify the capabilities of the parametric equaliser if supported on a receiver and/or transmitter. It is accessed from the *lpParaEqCaps* field of the RADIODEVCAPS structure.

C/C++:

```
typedef struct _PARAEQCAPS {
   int    iMaxParaPoles;
   int    iMinParaFreq;
   int    iMaxParaFreq;
   int    iMinParaQ;
   int    iMaxParaQ;
   int    iParaLevelRange;
   int    iParaLevelStep;
} PARAEQCAPS, FAR *LPPARAEQCAPS;
```

Delphi:

```
type
   PParaEqCaps = ^TParaEqCaps;
   TParaEqCaps = record
       iMaxParaPoles: Integer;
       iMinParaFreq: Integer;
       iMaxParaFreq: Integer;
       iMinParaQ: Integer;
       iMaxParaQ: Integer;
       iParaLevelRange: Integer;
       iParaLevelStep: Integer;
   end;
```

Fields

iMaxParaPoles

Specifies the maximum number of poles the parametric equaliser supports.

iMinParaFreq

Specifies the minimum frequency for a pole in Hz.

iMaxParaFreq

Specifies the maximum frequency for a pole in Hz.

iMinParaQ

Specifies the minimum Q for a pole in increments of 0.1.

iMaxParaQ

Specifies the maximum Q for a pole in increments of 0.1.

iParaLevelRange

Specifies the maximum level adjustment range (boost or cut) of a pole.

If `RADIOCAL_EQUALIZER` is specified in the *dwCalibrated* field of the `RADIODEVCAPS` structure, this value is in multiples of 0.1 dB.

iParaLevelStep

Specifies the granularity of the level adjustment.

PARAEQPARAMS

The `PARAEQPARAMS` structure is used to define a node in a parametric equaliser. This is used in the `PMR/T_AUDIOFILTER` command and the `PNR/T_AUDIOFILTER` notification. The *lpData* parameter can point to an array of these (the number determined by the *cbData* parameter).

A parametric equaliser provides the ability to adjust the centre frequency, Q and gain of a number of independent poles (defined by *iMaxParaPoles* in the `RADIODEVCAPS` structure), in order to compensate for non-ideal room acoustics. The common graphic equaliser (which also may be available in the `AUDIOFILTER` command and notification) is a form of parametric equaliser, in which the pole frequencies and Qs are fixed and only the gain of each pole is variable.

C/C++:

```
typedef struct _PARAEQPARAMS {
   DWORD  dwFreq;
   DWORD  dwQ;
   DWORD  dwLevel;
} PARAEQPARAMS, FAR *LPPARAEQPARAMS;
```

Delphi:

```
type
   PParaEqParams = ^TParaEqParams;
   TParaEqParams = record
       dwFreq: Longint;
       dwQ: Longint;
       dwLevel: Longint;
   end;
```

Fields

dwFreq

Specifies the centre frequency of the gain or attenuation in Hz. The frequency can range from 0 to *iMaxParaFreq* specified in the `RADIODEVCAPS` structure. *iMaxParaFreq* will typically not exceed 20 kHz.

dwQ

Specifies the 'Q' of each pole and equals the ratio of its centre frequency to its bandwidth at –3 dB. The value of Q is specifies in multiples of 0.1 and ranges from 0.1 to *iMaxParaQ* specified in the RADIODEVCAPS structure.

dwLevel

Specifies the amount of boost or cut at the centre frequency. A positive level is used to provide gain, and a negative level is used to provide a partial notch.

The limits of gain or attenuation are specified by the *iParaLevelRange* in the RADIODEVCAPS structure.

RADIODEVCAPS

The RADIODEVCAPS structure describes the capabilities of a radio device that is passed as a parameter in the xrsPluginStart function.

The SDK provides a function xrsCopyRadioDevCaps to make a copy of this structure for local use.

C/C++:

```
typedef struct _RADIODEVCAPS {
   int     cbTotalSize;
   int     cbFixedSize;
   int     cbFreqRangeSize;

   /*--- Product information ---*/
   CHAR    szManufacturer[32];
   CHAR    szProduct[32];
   CHAR    szSerialNum[16];
   CHAR    szUserDefName[64];
   DWORD   dwAppVersion;

   int     iDeviceNum;

   /*--- Global information ---*/
   DWORD   dwFreqRes;
   DWORD   dwCalibrated;

   LPTONECAPS      lpToneCaps;
   int     iMinBpFreq;
   int     iMaxBpFreq;
   LPPARAEQCAPS    lpParaEqCaps;
   LPGRAPHEQCAPS   lpGraphEqCaps;

   /*--- Receiver information ---*/
   DWORD   dwRxFeatures;

   LPVOID  lpRxExtraInfo;

   int     iSquelchFeatures;
   int     iMinSquelchLevel;
   int     iMaxSquelchLevel;
   int     iMinSquelchNoise;
   int     iMaxSquelchNoise;

   int     iNumRxFreqRanges;
   LPFREQRANGE     lpRxFreqRanges;

   int     iNumRxModes;
   int     cbDemodDefSize;
   LPDEMODDEF      lpRxModeDefs;

   int     iNumRfInputs;
   int     iMaxAtten;
   int     iAttenStep;
   int     iMaxPreamp;
   int     iPreampStep;
```

```
    int     iAgcSpeeds;
    LPAGCEXCAPS     lpAgcExCaps;

    int     iMinIfGain;
    int     iMaxIfGain;

    int     iMaxVolume;
    int     iVolumeStep;
    int     iBalanceRange;
    int     iBalanceStep;
    int     iRxAudioSources;

    int     iMaxNbThreshold;
    int     iMaxNotchFreq;
    int     iMaxNoiseReduction;

    LPDSPCAPS       lpRxDspCaps;

    /*--- Transmitter information ---*/
    DWORD   dwTxFeatures;
    LPVOID  lpTxExtraInfo;

    int     iNumTxFreqRanges;
    LPFREQRANGE     iTxFreqRanges;

    int     iNumTxModes;
    int     cbModDefSize;
    LPMODDEF        lpTxModeDefs;
    int     iTxModSources;

    int     iMaxTxPower;

    int     iMaxAntiVox;

    int     iAudioProcFlags;
    int     cbAudioProcSize;
    LPTXAUDIOPROC lpAudioProcCaps;

    int     iTxSelCallTypes;
    int     iMaxToneLevel;
    int     iMaxToneDuration;

    int     iTxInitiators;
    int     iTxMaxReleaseDelay;

    LPDSPCAPS       lpTxDspCaps;

    /*--- Memory Support ---*/
    DWORD   dwMemFeatures;
    DWORD   dwMaxRecords;
    int     iNumBanks;
} RADIODEVCAPS, FAR *LPRADIODEVCAPS;
```

Delphi:

```
type
    PRadioDevCaps = ^TRadioDevCaps;
    TRadioDevCaps = record
        cbTotalSize: Integer;
        cbFixedSize: Integer;
        cbFreqRangeSize: Integer;

        {--- Product information ---}
        szManufacturer: array [0..31] of Char;
        szProduct: array [0..31] of Char;
        szSerialNum: array [0..15] of Char;
        szUserDefName: array [0..63] of Char;
        dwAppVersion: Longint;
        iDeviceNum: Integer;

        {--- Global information ---}
```

```
    dwFreqRes: Longint;
    dwCalibrated: Longint;

    lpToneCaps: PToneCaps;
    iMinBpFreq: Integer;
    iMaxBpFreq: Integer;
    lpParaEqCaps: PParaEqCaps;
    lpGraphEqCaps: PGraphEqCaps;

    {--- Receiver information ---}
    dwRxFeatures: Longint;
    lpRxExtraInfo: Pointer;

    iSquelchFeatures: Integer;
    iMinSquelchLevel: Integer;
    iMaxSquelchLevel: Integer;
    iMinSquelchNoise: Integer;
    iMaxSquelchNoise: Integer;

    iNumRxFreqRanges: Integer;
    lpRxFreqRanges: PFreqRange;

    iNumRxModes: Integer;
    cbDemodDefSize: Integer;
    lpRxModeDefs: PDemodDef;

    iNumRfInputs: Integer;
    iMaxAtten: Integer;
    iAttenStep: Integer;
    iMaxPreamp: Integer;
    iPreampStep: Integer;

    iAgcSpeeds: Integer;
    lpAgcExCaps: PAgcExCaps;

    iMinIfGain: Integer;
    iMaxIfGain: Integer;

    iMaxVolume: Integer;
    iVolumeStep: Integer;
    iBalanceRange: Integer;
    iBalanceStep: Integer;
    iRxAudioSources: Integer;

    iMaxNbThreshold: Integer;
    iMaxNotchFreq: Integer;
    iMaxNoiseReduction: Integer;

    lpRxDspCaps: PDspCaps;

    {--- Transmitter information ---}
    dwTxFeatures: Longint;
    lpTxExtraInfo: Pointer;

    iNumTxFreqRanges: Integer;
    lpTxFreqRanges: PFreqRange;

    iNumTxModes: Integer;
    cbModDefSize: Integer;
    lpTxModeDefs: PModDef;
    iTxModSources: Integer;

    iMaxTxPower: Integer;

    iMaxAntiVox: Integer;

    iAudioProcFlags: Integer;
    cbAudioProcSize: Integer;
    lpAudioProcCaps: PTxAudioProc;

    iTxSelCallTypes: Integer;
```

```
        iMaxToneLevel: Integer;
        iMaxToneDuration: Integer;

        iTxInitiators: Integer;
        iTxMaxReleaseDelay: Integer;

        lpTxDspCaps: PDspCaps;

        {--- Memory support ---}
        dwMemFeatures: Longint;
        dwMaxMemories: Longint;
        iNumBanks: Integer;
    end;
```

Fields

cbTotalSize

Specifies the total size of entire RADIODEVCAPS data including any variable length data located after the fixed size structure (the size specified by *cbFixedSize*). If a copy of this structure is required, this specifies the amount of data to copy. Any pointers in the structure have to be translated when copied. The SDK provides a function xrsCopyRadioDevCaps to simplify this process.

cbFixedSize

Specifies the size of this structure. This may change in the future as the size may grow as more features are supported.

cbFreqRangeSize

Specifies the size of a FREQRANGE structure that is used in an array specifying all supported receiver and transmitter frequency ranges or bands.

szManufacturer

A null-terminated string that specifies the manufacturer of the radio device.

szProduct

A null-terminated string that specifies the model number of the radio device.

szSerialNum

A null-terminated string that specifies the product's serial number. This is manufacturer and product specific.

szUserDefName

A null-terminated string that the user defines for the device.

dwAppVersion

Specifies the application version. The high word contains the major version number, the low word specifies the minor version number of the application.

iDeviceNum

A logical device number for the radio device. This can be used to uniquely identify each device in a multi-device system.

dwFreqRes

Specifies the frequency resolution of the device in Hz. This applies to both receivers and transmitters.

dwCalibrated

This specifies a range of flags to indicate which features are expressed in actual units instead of arbitrary values. The flags include:

RADIOCAL_SLEVEL ..... *PN_SLEVEL and squelch level notifications in* ***dBm***
RADIOCAL_ATTEN ..... *attenuator notifications, commands & capabilities in* ***dB***
RADIOCAL_PREAMP ..... *preamp notifications, commands & capabilities in* ***dB***
RADIOCAL_IFGAIN ..... *IF gain notifications, commands & capabilities in* ***dB***

RADIOCAL_VOLUME ..... *volume notifications, commands & capabilities in **0.1 dB** steps*
RADIOCAL_BALANCE ..... *balance notifications, commands & capabilities in **0.1 dB** steps*
RADIOCAL_TONE ..... *bass, treble (& mid) in **0.1 dB** steps*
RADIOCAL_EQUALIZER ..... *parametric and/or graphic equaliser levels in **0.1 dB** steps*

RADIOCAL_TXPOWER ..... *transmitter power notifications, commands & capabilities in **mW***
RADIOCAL_ANTIVOX ..... *anti-vox notifications, commands & capabilities in **dB***
RADIOCAL_AUDIOGAIN ..... *audio gain notifications, commands & capabilities in **dB***
RADIOCAL_SSBMODPEP ..... *LSB and USB 'peak envelope power' expressed as a **%***
RADIOCAL_AMMODDEPTH ..... *AM modulation depth notifications, commands & cap's in **%***
RADIOCAL_FMDEV ..... *FM deviation notifications, commands & capabilities in **Hz***
RADIOCAL_FMWPILOTTONE ..... *FM pilot tone value in **Hz***

lpToneCaps

Points to a TONECAPS structure that specifies the tone (bass, treble and mid-range) adjustment capabilities of the receiver's audio output and/or the transmitter's input. If the device doesn't support tone controls, this is NULL.

iMinBpFreq

Specifies the minimum frequency in Hz that can be set for the audio bandpass filter in the receiver's audio output and/or the transmitter's input. This is only used if the RADIOCAPS_BPFILTER flag is set in the *dwRxFeatures* and/or *dwTxFeatures* fields.

iMaxBpFreq

Specifies the maximum frequency in Hz for the audio bandpass filter(s) in the device.

lpParaEqCaps

Points to a PARAEQCAPS structure that specifies the capabilities of the parametric equaliser if the device has one. The RADIOCAPS_PARAMETRIC flag set in the *dwRxFeatures* and/or *dwTxFeatures* fields specifies support.

lpGraphEqCaps

Points to a GRAPHEQCAPS structure that specifies the capabilities and properties of the graphic equaliser if the device supports it. Support is specified by the RADIOCAPS_EQUALIZER flag set in the *dwRxFeatures* and/or *dwTxFeatures* fields.

dwRxFeatures

This specifies a range of flags to indicate which features the receiver supports. The feature flags include:

RADIOCAPS_RECEIVER ..... *supports radio reception*
RADIOCAPS_POWER ..... *supports on/off power control*
RADIOCAPS_EXTREFOSC ..... *supports external reference osc. input*

RADIOCAPS_BASSTREBLE ..... *supports base/treble tone controls*
RADIOCAPS_MIDRANGE ..... *supports mid-range tone control*
RADIOCAPS_BPFILTER ..... *supports bandpass filter*
RADIOCAPS_PARAMETRIC ..... *supports parametric equalizer*
RADIOCAPS_EQUALIZER ..... *supports graphic equalizer*

RADIORXCAPS_PREAMP ..... *supports controllable preamp*
RADIORXCAPS_ATTEN ..... *supports controllable attenuator*

RADIORXCAPS_AGC ..... *supports switchable AGC (on/off/speed)*
RADIORXCAPS_ADJAGC ..... *supports adjustable AGC parameters (attack, hold & decay)*
RADIORXCAPS_IFGAIN ..... *supports adjustable IF gain*
RADIORXCAPS_AGCGAIN ..... *supports adjustable maximum AGC gain*
RADIORXCAPS_AFC ..... *supports switchable AFC*

RADIORXCAPS_FMWSTEREO ..... *supports stereo reception in FMW*
RADIORXCAPS_STEREO ..... *supports stereo reception in other modes*

RADIORXCAPS_BALANCE ..... *supports audio balance control*
RADIORXCAPS_LOUD ..... *supports switchable loudness compensation*

`RADIORXCAPS_NOISEBLANKER` ..... *supports noise blanker*
`RADIORXCAPS_AUTONOTCH` ..... *supports automatic notch filter*
`RADIORXCAPS_MANUALNOTCH` ..... *supports manual notch filter*
`RADIORXCAPS_NOISEREDUCTION` ..... *supports noise reduction*

`RADIORXCAPS_BLOCKSCAN` ..... *supports the PM_BLOCKSCAN command*
`RADIORXCAPS_TRUNKING` ..... *supports trunking decoding and tracking*

lpRxExtraInfo

Pointer to extra info about the capabilities of the receiver. This is NULL if there is no extra information. If it is not NULL, the first integer that this points to specifies the amount of extra information supplied in bytes.

iSquelchFeatures

Specifies which squelch features are supported:

`RXSQUELCH_SLEVEL` ..... *supports squelch by signal level*
`RXSQUELCH_NOISE` ..... *supports squelch by noise level*
`RXSQUELCH_CTCSS` ..... *supports squelch by CTCSS tone*
`RXSQUELCH_SYLLABIC` ..... *supports squelch by syllabic content*
`RXSQUELCH_DTMF` ..... *supports squelch by DTMF tone burst*
`RXSQUELCH_2TONE` ..... *supports squelch by 2-tone burst*
`RXSQUELCH_5TONE` ..... *supports squelch by 5-tone burst*
`RXSQUELCH_DPL` ..... *supports squelch by DPL burst*

iMinSquelchLevel

Specifies the minimum signal level that can be set for squelch control.

If the `RADIOCAL_SLEVEL` flag is set in the *dwCalibrated* field, then this value is in dBm.

iMaxSquelchLevel

Specifies the maximum signal level that can be set for squelch control.

If the `RADIOCAL_SLEVEL` flag is set in the *dwCalibrated* field, then this value is in dBm.

iMinSquelchNoise

Specifies the minimum noise level that can be set for squelch control.

iMaxSquelchNoise

Specifies the maximum noise level that can be set for squelch control.

iNumRxFreqRanges

Specifies the number of defined receiver frequency ranges in the `FREQRANGE` array pointed to by the *lpRxFreqRanges* field below.

lpRxFreqRanges

Points to the receiver's supported frequency ranges in a `FREQRANGE` array. The *iNumRxFreqRanges* field above specifies the number of ranges in this array.

iNumRxModes

Specifies the number of defined receiver modes (some may be duplicates with different fixed IF bandwidths). All modes are stored in a `DEMODDEF` array pointed to by the *lpRxModeDefs* field.

cbDemodDefSize

Specifies the size of the `DEMODDEF` structure that is used in an array specifying all supports modes and associated properties.

lpRxModeDefs

Points to the receiver's supported demodulation modes in a `DEMODDEF` array. The *iNumRxModes* field specifies the number of modes in this array.

iNumRfInputs

Specifies how many RF inputs the receiver has.

iMaxAtten

Specifies the maximum RF attenuation of the receiver's attenuator.

iAttenStep

Specifies the granularity of the RF attenuator. If this is the same as *iMaxAtten* then the receiver only has an on/off attenuator. If this is one, the attenuator is continuously adjustable from zero to *iMaxAtten*.

iMaxPreamp

Specifies the maximum RF amplification level of the receiver.

iPreampStep

Specifies the granularity of the amplification level. If this is the same as *iMaxPreamp* then the receiver only has an on/off preamplifier. If this is one, the preamplifier is continuously adjustable from zero to *iMaxPreamp*.

iAgcSpeeds

Specifies a range of flags that specify generic AGC speeds that the receiver supports:

```
RXAGCCAPS_OFF
RXAGCCAPS_MEDIUM
RXAGCCAPS_SLOW
RXAGCCAPS_FAST
RXAGCCAPS_VSLOW
RXAGCCAPS_VFAST
```

Each set flag corresponds to a `RXAGC_xxx` constant that can be used in the `PMR_AGC` command.

lpAgcExCaps

Points to an `AGCEXCAPS` structure that contains extended AGC capabilities of the receiver. If the receiver doesn't support these capabilities, then this is NULL.

iMinIfGain

Specifies the minimum IF gain level. This can be below 0 signifying the receiver also supports IF attenuation. This field is only valid when the `RADIORXCAPS_IFGAIN` flag is set in the *dwRxFeatures* field.

iMaxIfGain

Specifies the maximum IF gain level.

iMaxVolume

Specifies the maximum volume level that can be set. The lowest volume is always zero.

iVolumeStep

Specifies the granularity of the volume control. If this is one or zero, the volume is continuously adjustable from zero to *iMaxVolume*.

iBalanceRange

Specifies the maximum absolute value (positive or negative) for the `PMR_BALANCE` command. This is only supported if the `RADIORXCAPS_BALANCE` flag is set in the *dwRxFeatures* field.

iBalanceStep

Specifies the granularity for balance adjustment.

iRxAudioSources

Specifies a range of flags that specify supported selectable audio sources for the device's audio output:

`RXAUDIOSRCCAPS_RADIO` ..... *receiver demodulator*
`RXAUDIOSRCCAPS_EXT` ..... *external (line in)*

RXAUDIOSRCCAPS_DSP ..... *DSP/DAC*

Each set flag corresponds to a RXAUDIOSRC_xxx constant that can be used in the PMR_AUDIOSRC command.

iMaxNbThreshold

Specifies the maximum noise blanker threshold that can be set in the PMR_NOISEBLANKER command. This is supported if the RADIORXCAPS_NOISEBLANKER flag is set in the *dwRxFeatures* field.

iMaxNotchFreq

Specifies the maximum frequency in Hz that the manual notch filter can be set to. Support for a manual notch is specified the RADIORXCAPS_MANUALNOTCH flag set in the *dwRxFeatures* field.

iMaxNoiseReduction

Specifies the maximum noise reduction type that can be selected with the PMR_NOISEREDUCT command. This is supported if the RADIORXCAPS_NOISEREDUCT flag is set in the *dwRxFeatures* field.

lpRxDspCaps

Points to a DSPCAPS structure that specifies the capabilities of the DSP in the receiver. This is NULL if the receiver does not have a DSP that can be controlled by a plug-in.

dwTxFeatures

This specifies a range of flags where each flag that is set indicates a particular feature the transmitter supports. The feature flags include:

RADIOCAPS_TRANSMITTER ..... *supports radio transmitting*
RADIOCAPS_POWER ..... *supports on/off power control*
RADIOCAPS_EXTREFOSC ..... *supports external reference osc. input*

RADIOCAPS_BASSTREBLE ..... *supports base/treble tone controls*
RADIOCAPS_MIDRANGE ..... *supports mid-range tone control*
RADIOCAPS_BPFILTER ..... *supports bandpass filter*
RADIOCAPS_PARAMETRIC ..... *supports parametric equalizer*
RADIOCAPS_EQUALIZER ..... *supports graphic equalizer*

RADIOTXCAPS_SUBCARRIER ..... *supports sub-carrier transmission*
RADIOTXCAPS_ANTIVOX ..... *supports anti-vox adjustment*
RADIOTXCAPS_AUDIOGAIN ..... *supports audio input gain adjustment*
RADIOTXCAPS_TXRELEASE ..... *supports adjustable Tx release times*
RADIOTXCAPS_ADJDTMFBURST ..... *supports adjustable DTMF burst duration*
RADIOTXCAPS_ADJTONERATE ..... *supports adjustable 2/5-tone rate*
RADIOTXCAPS_FMWSTEREO ..... *supports stereo transmission in FMW*
RADIOTXCAPS_STEREO ..... *supports stereo transmission in other modes*

lpTxExtraInfo

Pointer to extra info about the capabilities of the transmitter. This is NULL if there is no extra information. If it is not NULL, the first integer that this points to specifies the amount of extra information supplied in bytes.

iNumTxFreqRanges

Specifies the number of defined transmitter frequency bands in the FREQRANGE array pointed to by the *lpTxFreqRanges* field below.

lpTxFreqRanges

Points to the transmitter's supported frequency bands in a FREQRANGE array. The *iNumTxFreqRanges* field above specifies the number of bands in this array.

iNumTxModes

Specifies the number of supported transmitter modes. All modes are stored in a MODDEF array pointed to by the *lpTxModeDefs* field.

cbModDefSize

Specifies the size of the MODDEF structure that is used in an array specifying all supports modes and associated properties.

lpTxModeDefs

Points to the transmitter's supported modulation modes in a MODDEF array. The *iNumTxModes* field specifies the number of modes in this array.

iTxModSources

Contains a range of flags that specify supported sources to the transmitter's demodulator:

TXMODSRCCAPS_MIC ..... *supports microphone source*
TXMODSRCCAPS_EXT ..... *supports external audio signal source*
TXMODSRCCAPS_DSP ..... *supports source from computer from DAC and/or DSP*
TXMODSRCCAPS_KEY ..... *supports morse key source*
TXMODSRCCAPS_MISC1 ..... *supports miscellaneous source*
TXMODSRCCAPS_MISC2 ..... *supports another miscellaneous source*

Each set flag corresponds to a TXMODSRC_xxx constant that can be used in the PMT_MODSRC command.

iMaxTxPower

Specifies the maximum transmitter output power that can be set with the PMT_RFPOWER command.

iMaxAntiVox

Specifies the maximum anti-vox level that can be set with the PMT_ANTIVOX command.

iAudioProcFlags

Contains a range of flags that specify which audio input processing the transmitter supports. These include:

TXAUDIOPROC_COMP ..... *supports compression*
TXAUDIOPROC_CLIP ..... *supports clipping*
TXAUDIOPROC_AGC ..... *supports AGC*

cbAudioProcSize

Specifies the size of the TXAUDIOPROC structure referred to by the *lpAudioProcCaps* field below.

lpAudioProcCaps

Points to a TXAUDIOPROC structure that specifies the maximum values for each of the supported audio processing features. If any feature is not supported, its associated field is set to zero.

iTxSelCallTypes

Specifies which selective calling types are supported by the transmitter with an set of flags:

TXSELCALLCAPS_NORMAL ..... *supports no selective calling*
TXSELCALLCAPS_CTCSS ..... *supports CTCSS*
TXSELCALLCAPS_SINGLE ..... *supports single tone burst*
TXSELCALLCAPS_DTMF ..... *supports DTMF burst*
TXSELCALLCAPS_2TONE ..... *supports two-tone sequential burst*
TXSELCALLCAPS_5TONE ..... *supports five-tone sequential burst*
TXSELCALLCAPS_DPL ..... *supports DPL burst*

Each set flag corresponds to a TXSELCALL_xxx constant that can be used in the PMT_SELCALL command.

iMaxToneLevel

Specifies the maximum tone level that can be set for selective calling types that use tones.

iMaxToneDuration

Specifies the maximum duration that can be set for a tone burst in selective calling types that use a tone burst.

iTxInitiators

Contains a range of flags that specify the supported methods of transmission activation:

`TXINITIATE_MICSWITCH` ..... *manual activation by microphone switch*
`TXINITIATE_SECONDARY` ..... *manual activation by secondary switch (eg. foot-switch)*
`TXINITIATE_SOFTWARE` ..... *manual activation by software (see PMT_TX command)*
`TXINITIATE_VOX` ..... *voice activation*

iTxMaxReleaseDelay

Specifies the maximum release delay that can be set for a transmitter.

lpTxDspCaps

Points to a `DSPCAPS` structure that specifies the capabilities of the DSP in the input section of the transmitter. This is NULL if the transmitter does not have a DSP that can be controlled by a plug-in.

dwMemFeatures

Specifies which fields are supported by the frequency memory in the application:

`RADIOMEM_BANKS` ..... *memory allocated in banks*
`RADIOMEM_FOLDERS` ..... *stores entries in folders with tree structure and text descriptions*

`RADIOMEM_NAME` ..... *includes name field*
`RADIOMEM_MODE` ..... *includes mode field*
`RADIOMEM_SQUELCH` ..... *includes squelch settings*
`RADIOMEM_STEPSIZE` ..... *includes step size to set when recalled*
`RADIOMEM_GROUPS` ..... *includes group allocations*
`RADIOMEM_ATTEN` ..... *includes attenuator settings*
`RADIOMEM_PREAMP` ..... *includes preamp settings*
`RADIOMEM_BANDWIDTH` ..... *includes bandwidth settings*
`RADIOMEM_IFSHIFT` ..... *includes IF shift settings*
`RADIOMEM_AGC` ..... *includes AGC settings (including IF gain)*
`RADIOMEM_AFC` ..... *includes AFC settings*
`RADIOMEM_HITCOUNT` ..... *includes number of signal hits*
`RADIOMEM_SLEVEL` ..... *includes signal level readings (max and/or last)*
`RADIOMEM_SCHEDULE` ..... *includes transmission schedule*
`RADIOMEM_DATETIME` ..... *includes date/time stored, modified and/or accessed*
`RADIOMEM_LOCKOUT` ..... *includes memory lockout flag*
`RADIOMEM_CALLSIGN` ..... *includes station callsign*
`RADIOMEM_COMMENT` ..... *includes comments/description*

dwMaxRecords

Specifies the maximum number of records that can be stored in each bank or folder. For applications that do not support banks or folders, this specifies the maximum number that can be stored in a file. If this is set to zero, there is no practical limit.

iNumBanks

Specifies the number of banks in the application's frequency memory. If the application does not support banks, this is either set to zero or one.

SQUELCHSETTINGS

The `SQUELCHSETTINGS` structure is used to define the squelch parameters for enabled squelch features. This is used in the `PMR_SQUELCH` command and the `PNR_SQUELCH` notification.

C/C++:

```
typedef struct _SQUELCHSETTINGS {
   DWORD  dwSLevel;
   DWORD  dwNLevel;
   DWORD  dwCtcssFreq;
   DWORD  dwBurstType;
   DWORD  dwBurstData;
} SQUELCHSETTINGS, FAR *LPSQUELCHSETTINGS;
```

Delphi:

```
type
   PSquelchSettings = ^TSquelchSettings;
   TSquelchSettings = record
       dwSLevel: Longint;
       dwNLevel: Longint;
       dwCtcssFreq: Longint;
       dwBurstType: Longint;
       dwBurstData: Longint;
   end;
```

Fields

dwSLevel

If signal level squelch is enabled and the signal level is below *dwSLevel*, the squelch will activate. The value must be between *iMinSquelchLevel* and *iMaxSquelchLevel* as defined in the RADIODEVCAPS structure.

dwNLevel

If noise squelch is enabled and the noise level is above *dwNLevel*, the squelch will activate. The value must be between *iMinSquelchNoise* and *iMaxSquelchNoise* as defined in the RADIODEVCAPS structure.

dwCtcssFreq

If CTCSS squelch is enabled and the CTCSS tone received is different to *dwCtcssFreq*, the squelch will activate. The tone frequency is specified in milli-hertz (mHz).

dwBurstType

Can be one of the following burst types:

- DTMF burst (inactive / **tone-pair**)
- 2-tone burst (inactive / **tone-set**)
- 5-tone burst (inactive / **tone-set**)
- DPL burst (inactive / **address** and **EOT**)

If enabled, the squelch will deactivate when the tone burst is received (it will reactive when one of the previous conditions are met).

dwBurstData

Specifies the data for the above burst type (shown in bold).

Remarks

At least one of the following squelch methods has to be enabled:

- Signal level (inactive / **s-level** = *iMinSquelchLevel* to *iMaxSquelchLevel*)
- Noise (inactive / **n-level** = *iMinSquelchNoise* to *iMaxSquelchNoise*)
- CTCSS (inactive / **freq** in mHz)
- Syllabic (inactive / active)

and optionally one of:

- DTMF burst (inactive / **tone-pair**)
- 2-tone burst (inactive / **tone-set**)
- 5-tone burst (inactive / **tone-set**)
- DPL burst (inactive / **address** and **EOT**)

Squelch activity can be described with the following formulae:

Inactive = (SLevel ≥ s-level) and (Noise ≤ n-level) and (CTCSS = freq) and (Syllabic) and
{(DTMF = tone-pair) or (*n*-tone = tone-set) or (DPL = address)}

Active = (SLevel < s-level) or (Noise > n-level) or (CTCSS ≠ freq) or (not Syllabic) or (DPL = EOT)

where ( ) = equate if enabled, otherwise ignore

TONECAPS

The TONECAPS structure is used in the RADIODEVCAPS structure to specify the capabilities of the audio tone adjustments in the receiver and/or transmitter.

If the RADIOCAL_TONE flag is specified in the *dwCalibrated* field of the RADIODEVCAPS structure, then all the values in this structure are multiples of 0.1 dB.

C/C++:

```
typedef struct _TONECAPS {
   int    iBassRange;
   int    iBassStep;
   int    iMidRange;
   int    iMidStep;
   int    iTrebleRange;
   int    iTrebleStep;
} TONECAPS, FAR *LPTONECAPS;
```

Delphi:

```
type
   PToneCaps = ^TToneCaps;
   TToneCaps = record
       iBassRange: Integer;
       iBassStep: Integer;
       iMidRange: Integer;
       iMidStep: Integer;
       iTrebleRange: Integer;
       iTrebleStep: Integer;
   end;
```

Fields

iBassRange

Specifies the maximum range of bass adjustment above and below the normal level.

iBassStep

Specifies the granularity of bass adjustment.

iMidRange

Specifies the maximum range of mid-range adjustment above and below the normal level.

iMidStep

Specifies the granularity of mid-range adjustment.

iTrebleRange

Specifies the maximum range of treble adjustment above and below the normal level.

iTrebleStep

Specifies the granularity of treble adjustment.

TXAUDIOPROC

The TXAUDIOPROC structure is used to specify the parameters for audio input processing on a transmitter. It also used in the *lpAudioProcCaps* field in the RADIODEVCAPS structure for specifying the maximum values for these parameters.

C/C++:

```
typedef struct _TXAUDIOPROC {
   DWORD dwCompression;
   DWORD dwClipping;
   DWORD dwAgc;
} TXAUDIOPROC, FAR *LPTXAUDIOPROC;
```

Delphi:

```
type
   PTxAudioProc = ^TTxAudioProc;
   TTxAudioProc = record
       dwCompression: Longint;
       dwClipping: Longint;
       dwAgc: Longint;
   end;
```

Fields

dwCompression

Specifies the compression level from above zero to *dwCompression* specified under the *lpAudioProcCaps* field in the RADIODEVCAPS structure. If zero is specified, the signal is not compressed.

dwClipping

Specifies the clipping level from above zero to *dwClipping* specified under the *lpAudioProcCaps* field in the RADIODEVCAPS structure. If zero is specified, the signal is not clipped.

dwAgc

Specifies the AGC level from above zero to *dwAgc* specified under the *lpAudioProcCaps* field in the RADIODEVCAPS structure. If zero is specified, AGC is not applied to the signal.

XRS Commands

If memory is allocated for the *lpData* parameter of a command, after the command has returned, the memory can be immediately freed. This applies to all commands.

There are four classes of commands: ones that apply only to receivers (`PMR_xxx`), ones that apply only to transmitters (`PMT_xxx`), global and those that apply to neither (`PM_xxx`).

To determine whether the device supports receiving, transmitting or both, check for `RADIOCAPS_RECEIVER` and/or `RADIOCAPS_TRANSMITTER` in the *dwRxFeatures* and *dwTxFeatures* fields in the `RADIODEVCAPS` structure.

PM_CAPABILITIES

The PM_CAPABILITIES command informs the application that the capabilities of the receiver changes due to the plug-in starting/stopping. If the command is sent while the plug-in starting phase, the new capabilities must be specified through a modified copy of the RADIODEVCAPS structure passed as argument of the xrsPluginStart exported entry point. Any changes in the content of the structure should affect only sections covered by the running plug-in (i.e. only the list of available modes when the plug-in is a demodulator one). Any change to the capabilities of the radio receiver must be changed back when the plug-in is stopped.

Parameters

dwParam

Not used.

cbData

The amount of memory occupied by the new RADIODEVCAPS structure.

lpData

Pointer to the new RADIODEVCAPS. After passing the information to the XRS server the memory can be freed.

PM_CLOSED

The `PM_CLOSED` command informs the application that the plug-in has shut-down. The application will then stop sending notifications to the plug-in (but the plug-in can be started again at a later time by the application).

Parameters

dwParam

Not used.

lpData

Pointer to the null-terminated string that was sent to the application in the `xrsPluginInit` command. The strings must be exactly the same.

Return Value

Always zero.

PM_CREATEFOLDER

The `PM_CREATEFOLDER` command is sent to the application to create a new subfolder in the currently active folder.

Support for folders is specified by the presence of the `RADIOMEM_FOLDERS` flag in the *dwMemFeatures* field in the `RADIODEVCAPS` structure.

Parameters

dwParam

Not used.

lpData

Points to a null-terminated string that specifies the name of the folder to create. This cannot be the same as an existing subfolder in the currently active folder (but can be the same as subfolder in another folder).

Return Value

Zero if sub-folder was successfully created, otherwise `PLUGIN_CB_FAIL` is returned (0x80000000).

PM_DELETEFOLDER

The `PM_DELETEFOLDER` command is sent to the application to delete a subfolder in the currently active folder.

Support for folders is specified by the presence of the `RADIOMEM_FOLDERS` flag in the *dwMemFeatures* field in the `RADIODEVCAPS` structure.

Parameters

dwParam

Not used.

lpData

Points to a null-terminated string that specifies the name of the folder to delete.

Return Value

Zero if subfolder was successfully deleted, otherwise `PLUGIN_CB_FAIL` is returned (0x80000000).

PM_DISABLE

The `PM_DISABLE` command is sent to the application to disable various parts of the user interface.

Parameters

dwParam

`PD_NONE`	-	Enables everything
`PD_ALL`	-	Disables entire interface
`PD_POWER`	-	Disables the power on/off control
`PD_CLOSE`	-	Stops the interface/application from being closed
`PD_ACCESSORIES`	-	Disables miscellaneous accessories (for example, a spectrum analyser)
`PD_SCHEDULER`	-	Disables the task scheduler if it exists
`PD_PLUGINS`	-	Disables other plug-ins from being started
`PD_RXALL`	-	Disables all receiver controls
`PD_RXFREQ`	-	Disables receiver frequency setting controls
`PD_RXMODE`	-	Disables mode and IF bandwidth controls
`PD_RXSLEVEL`	-	Disables the signal meter
`PD_RXSQUELCH`	-	Disables squelch controls
`PD_RXRFINPUT`	-	Disables RF input selection controls
`PD_RXRFGAIN`	-	Disables attenuator and preamplifier controls
`PD_RXIFGAIN`	-	Disables AGC and IF gain controls
`PD_RXIFSHIFT`	-	Disables IF shift and/or BFO offset controls
`PD_RXAFC`	-	Disables the AFC control

PD_RXAUDIO	-	Disables audio processing controls
PD_RXEXTOSC	-	Disables the external oscillator control
PD_RXMEMORY	-	Disables memory controls
PD_RXSCANNER	-	Disables scanner controls
PD_RXDSP	-	Disables receiver DSP, record and playback controls
PD_TXALL	-	Disables all transmitter controls
PD_TXFREQ	-	Disables transmitter frequency setting controls
PD_TXMODE	-	Disables transmitter modulation controls
PD_TX	-	Disables the transmitter activation control
PD_TXINPUT	-	Disables transmitter input processing controls
PD_TXPOWER	-	Disables transmitter power controls
PD_TXSETTINGS	-	Disables transmission type controls
PD_TXEXTOSC	-	Disables the external oscillator control
PD_TXDSP	-	Disables transmitter DSP, record and playback controls

lpData

Not used.

Return Value

Zero if the command was successful, otherwise PLUGIN_CB_FAIL is returned (0x80000000).

See Also

PN_DISABLED

PM_EVENT

The PM_EVENT command is sent to the application to notify that a plug-in defined event has occurred.

Parameters

dwParam

A plug-in defined value that the application may utilise.

lpData

Pointer to a plug-in defined string that the application may utilise.

Return Value

Zero if the event was handled, otherwise PLUGIN_CB_FAIL is returned (0x80000000).

Remarks

The most common usage for this command is for automatically scheduled operations that rely on certain events occurring. The application can trigger a task when a specified value (from *dwParam*) is received or the value is within a specified range. Alternatively, the application can use the string value (from *lpData*) and perform an application specified string operation and if the operation is successful, initiate a task.

A plug-in write that utilises this command should fully document what values and/or strings are sent for events that it may issue so a user is able to set up a schedule to utilise these events. The documentation must be included in the plug-in's help file.

PM_FILTERFLAGS

The PM_FILTERFLAGS command is sent to the application to inform it that the plug-in notification filtering is to be changed. Notifications can be added or removed with this command.

Note: This is a software command and is not related to frequency domain filtering.

Parameters

dwParam

PNF_xxx flags. See xrsPluginStart for the list of flags.

lpData

Not used.

Return Value

Zero if the command was successful, otherwise PLUGIN_CB_FAIL is returned (0x80000000).

PM_GETMEM

The PM_GETMEM command is sent to the application to retrieve the contents of a memory record in the currently active band or folder (if applicable).

Parameters

dwParam

The record number to retrieve.

lpData

Pointer to a MEMORYENTRY structure that will be filled with the memory record.

Return Value

Zero if the memory record was successfully retrieved, otherwise PLUGIN_CB_FAIL is returned.

PM_GETMEMFILE

The PM_GETMEMFILE command is sent to the application to retrieve the name of the active memory file.

Parameters

dwParam

Not used.

lpData

Pointer to a buffer to receive the null-terminated memory file name.

Return Value

Zero if the file name was successfully retrieved, otherwise PLUGIN_CB_FAIL is returned (0x80000000).

See Also

PM_SETMEMFILE

PN_MEMFILE

PM_GETNEXTFOLDER

The PM_GETNEXTFOLDER command is sent to the application to obtain the next subfolder name after the specified folder name. Neither folder has to be active.

Support for folders is specified by the presence of the RADIOMEM_FOLDERS flag in the *dwMemFeatures* field in the RADIODEVCAPS structure.

Parameters

dwParam

Not used.

lpData

Pointer to a buffer that contains the null-terminated folder name and has to be large enough to receive the next folder's name. If the current folder is an empty string, it will retrieve the first subfolder in the root of the memory.

Refer to PM_OPENFOLDER for details on folder names.

Return Value

Length of the folder name if successful, otherwise PLUGIN_CB_FAIL is returned (0x80000000).

See Also

PM_GETSUBFOLDER

PM_GETNEXTMEM

The PM_GETNEXTMEM command is sent to the application to obtain the next memory record number that contains information. It can also return the contents of the record.

Parameters

dwParam

Specifies the record number where the command returns the next available number (ie. record that contains data). If this is –1, the command returns the first record number that exists in the memory.

lpData

Can be NULL if the contents of the record is not desired.

If it is not NULL, points to a MEMORYENTRY structure that will receive the contents of the next memory record.

Return Value

If successful, returns the next memory record number that contains information. If there are no more records then –1 is returned.

PLUGIN_CB_FAIL is returned (0x80000000) if the command fails.

PM_GETNEXTPLUGIN

The PM_GETNEXTPLUGIN command is sent to the application to obtain a list of installed and running plug-ins in the application.

Parameters

dwParam

Not used.

lpData

Points to a buffer that contains a plug-in name of which the next plug-in will be returned. If the contents of buffer is just a null-terminator (0), the first plug-in installed will be returned in this buffer. The buffer should be at least 64 bytes in size to receive the name.

Return Value

If successful, the plug-in type (see `xrsPluginInit` for plug-in types) and if bit 16 is set (0x10000) the plug-in is currently running. If the command fails, `PLUGIN_CB_FAIL` is returned (0x80000000).

PM_GETNUMMEMS

The `PM_GETNUMMEMS` command is sent to the application to obtain the number of memory records that contains information.

Parameters

dwParam

Not used.

lpData

Not used.

Return Value

If successful, returns the number of records in the memory, otherwise `PLUGIN_CB_FAIL` is returned (0x80000000).

PM_GETSETTINGS

The `PM_GETSETTINGS` command is sent to the application to obtain one of the device's settings. This is typically used on start-up to obtain the device's settings (notifications are not sent to a plug-in after starting unless the setting changes) that the plug-in relies upon.

This function can be called from within the `xrsPluginStart` function (ie. before a handle to the plug-in is returned).

Parameters

dwParam

A notification code representing the setting to obtain. See the `PNR/T_xxx` notifications (not all notifications are supported, only those that involve device settings) for more information.

lpData

Points to a buffer that may be required depending on the notification code. If a notification requires a buffer but the size is unknown, this can be set to point to a single DWORD that will receive the size of the buffer required for full setting information.

This can be NULL if the notification doesn't use the *lpData* parameter.

Return Value

If successful, the setting according to the *dwParam* parameter of the notification, otherwise `PLUGIN_CB_FAIL` is returned (0x80000000).

PM_GETSUBFOLDER

The `PM_GETSUBFOLDER` command is sent to the application to obtain the name of the first subfolder in the specified folder (if one exists).

Support for folders is specified by the presence of the `RADIOMEM_FOLDERS` flag in the *dwMemFeatures* field in the `RADIODEVCAPS` structure.

Parameters

dwParam

Not used.

lpData

Pointer to a buffer that contains the null-terminated folder name and has to be large enough to receive the subfolder's name. If the current folder is an empty string, it will retrieve the first subfolder in the root of the memory.

Refer to PM_OPENFOLDER for details on folder names.

Return Value

Length of the folder name if successful, otherwise PLUGIN_CB_FAIL is returned (0x80000000).

See Also

PM_GETNEXTFOLDER

PM_MINIMIZE

The PM_MINIMIZE command is sent to the application to minimise or restore the application's user interface for the device.

Parameters

dwParam

If zero is specified, the interface is restored. If it is non-zero, the interface is minimised.

lpData

Not used.

See Also

PN_MINIMIZED

PM_MOVEFOLDER

The PM_MOVEFOLDER command is sent to the application to move the currently active folder and all its subfolders to another folder.

Support for folders is specified by the presence of the RADIOMEM_FOLDERS flag in the *dwMemFeatures* field in the RADIODEVCAPS structure.

Parameters

dwParam

Not used.

lpData

Pointer to a buffer that contains the null-terminated destination folder name. The current folder cannot be the root directory and the destination cannot be a subfolder of the active folder.

Refer to PM_OPENFOLDER for details on folder names.

Return Value

Zero if the command was successful, otherwise PLUGIN_CB_FAIL is returned (0x80000000).

PM_OPENFOLDER

The PM_OPENFOLDER command is sent to the application to set the active folder in memory.

Support for folders is specified by the presence of the RADIOMEM_FOLDERS flag in the *dwMemFeatures* field in the RADIODEVCAPS structure.

Parameters

dwParam

Not used.

lpData

Points to a null-terminated string that specifies the folder name to open. This can either be a relative path or an absolute path. Folders operate in a similar way to folders in many file-based operating systems.

Only the back-slash character cannot be used in a folder name (it is reserved for specifying a folder path).

A relative path is a folder name which must be a subfolder of the currently active folder (by default when a new memory file is opened, the active folder is set to the root). It can contain multiple subfolders where each folder name is separated by a back-slash.

An absolute path specifies all folder names from the root folder leading to the active folder, where each folder is separated by a back-slash and the first character is a back-slash (representing the root folder).

For example, 'Fire\Digital' is a relative path while '\Services\Emergency\Fire\Digital' is an absolute path.

Return Value

Zero if the folder was successfully opened, otherwise PLUGIN_CB_FAIL is returned (0x80000000).

See Also

PN_MEMFOLDER

PM_POWER

The PM_POWER command is sent to the application to control the device's power.

Support for this function is defined by presence of the RADIOCAPS_POWER flag in the *dwRxFeatures* or *dwTxFeatures* field in the RADIODEVCAPS structure.

Parameters

dwParam

If zero is specified, the power is switched off. If it is non-zero, the power is switched on.

lpData

Not used.

Return Value

Zero if the power state was successfully set, otherwise PLUGIN_CB_FAIL is returned (0x80000000).

See Also

PN_POWER

PM_RECALLMEM

The PM_RECALLMEM command is sent to the application to apply a memory record's settings to the device.

Parameters

dwParam

The memory record number to recall.

lpData

Not used.

Return Value

Zero if the memory record was successfully recalled, otherwise `PLUGIN_CB_FAIL` is returned.

See Also

`PN_MEMRECALL`

PM_SELECTBANK

The `PM_SELECTBANK` command is sent to the application to change the active memory bank.

Support for banks is specified by the presence of the `RADIOMEM_BANKS` flag in the *dwMemFeatures* field in the `RADIODEVCAPS` structure.

Parameters

dwParam

The bank number from 0 to the value specified by the *iNumBanks-1* field of the `RADIODEVCAPS` structure.

lpData

Not used.

Return Value

Zero if the bank was successfully selected, otherwise `PLUGIN_CB_FAIL` is returned (0x80000000).

See Also

`PN_MEMBANK`

PM_SETMEMFILE

The `PM_SETMEMFILE` command is sent to the application to load a different memory file.

Parameters

dwParam

Not used.

lpData

Points to a null-terminated string that specifies the memory file to load.

Return Value

Zero if the file was successfully loaded, otherwise `PLUGIN_CB_FAIL` is returned (0x80000000).

See Also

`PN_MEMFILE`

PM_GETMEMFILE

PM_STARTPLUGIN

The PM_STARTPLUGIN command can start another installed plug-in. To obtain a list of installed plug-ins (and their types), see the PM_GETNEXTPLUGIN command.

Parameters

dwParam

Not used.

lpData

Points to a null-terminated string that specifies the plug-in name to start.

Return Value

Zero if the plug-in was successfully started, or one if the plug-in was already running, otherwise PLUGIN_CB_FAIL is returned (0x80000000).

See Also

PN_PLUGINSTARTED

PM_STOPPLUGIN

The PM_STOPPLUGIN command can stop another running plug-in.

Parameters

dwParam

Not used.

lpData

Points to a null-terminated string that specifies the plug-in name to stop.

Return Value

Zero if the plug-in was successfully stopped, or one if the plug-in was not running, otherwise PLUGIN_CB_FAIL is returned (0x80000000).

See Also

PN_PLUGINSTOPPED

PM_STOREMEM

The PM_STOREMEM command is sent to the application the store the supplied settings into a memory record.

Parameters

dwParam

Specifies the memory record number to store the settings into.

lpData

Pointer to a MEMORYENTRY structure which contains the information to store into the memory.

Return Value

Zero if the record was successfully stored, otherwise PLUGIN_CB_FAIL is returned (0x80000000).

See Also

PN_MEMCHANGE

PM_VISIBLE

The PM_VISIBLE controls the whether the application's user interface for the device is hidden or shown.

Parameters

dwParam

Non-zero if the window is to be shown, zero if it is to be hidden.

lpData

Not used.

See Also

PN_VISIBLE

PMR/T_AUDIOFILTER

The PMR_AUDIOFILTER and PMT_AUDIOFILTER commands are sent to the application to control the receiver's audio output or transmitter's audio input filtering. Currently, four filter types are supported: bass, treble and optional mid-range tone controls; band-pass filter controls, parametric equaliser and graphic equaliser controls.

The supports for these controls are specified by the presence of the RADIOCAPS_BASSTREBLE (and RADIOCAPS_MIDRANGE), RADIOCAPS_BPFILTER, RADIOCAPS_PARAMETRIC and/or RADIOCAPS_EQUALIZER flags in the *dwRxFeatures* and *dwTxFeatures* field in the RADIODEVCAPS structure.

Parameters

dwParam

This can be one of the following filter types:

AUDIOFILTER_NONE	-	No filtering.
AUDIOFILTER_TONE	-	Bass, treble and optional mid-range tone filtering.
AUDIOFILTER_BANDPASS	-	Band-pass filtering.
AUDIOFILTER_PARAMETRIC	-	Parametric equaliser filtering.
AUDIOFILTER_GRAPHIC	-	Graphic equaliser filtering.

lpData

Depends on *dwParam*:

AUDIOFILTER_NONE:

Not used (can be NULL).

AUDIOFILTER_TONE:

Pointer to bass and treble DWORDs and a third mid-range level if supported. *cbSize* can equal eight or twelve depending on the presence of the mid-range value. The *iBassRange*, *iTrebleRange* and *iMidRange* fields in the TONECAPS structure define the range for each respectively. If the RADIOCAL_TONE flag is specified in the *dwCalibrated* field, then these values are specified in dB.

AUDIOFILTER_BANDPASS:

Pointer to a low-pass and high-pass frequency in Hz. Both these values are DWORDs and therefore *cbSize* must be set to eight. The *iMinBpFreq* and *iMaxBpFreq* fields of the RADIODEVCAPS structure specify the lowest and highest frequencies.

AUDIOFILTER_PARAMETRIC:

Pointer to an array of PARAEQPARAMS structures where for each entry a centre frequency, Q and level parameters are specified. The *iMaxParaPoles* field of the PARAEQCAPS structure defines the maximum number of poles that can be specified. The *iMinParaFreq* and *iMaxParaFreq* fields specify the frequency range for a pole. The *iMinParaQ* and *iMaxParaQ* fields specify the range of Q supported. The *iParaLevelRange* field defines the maximum level (above or below the nominal level), with the adjustment granularity specified by the *iParaLevelStep* field.

AUDIOFILTER_GRAPHIC:

Pointer to an array of DWORDS where for each equaliser frequency specified in the GRAPHEQCAPS structure, a corresponding level is specified. The *iNumFreqs* field specifies the number of frequencies that have to be set. The *iLevelRange* field defines the maximum level (above or below the nominal level), with the adjustment granularity specified by the *iLevelStep* field.

Return Value

Zero if the filter was successfully set, otherwise PLUGIN_CB_FAIL is returned (0x80000000).

See Also

PNR/T_AUDIOFILTER

PMR/T_DSPADCSTART

The PMR_DSPADCSTART and PMT_DSPADCSTART commands are sent to the application to initiate 'analog-digital conversion' of received audio signals. A typical application is to enable audio recording to a hard disk for playback at a later time.

This function can only be used if the RADIODSP_ADC flag is set in the *dwRxDspFeatures* and *dwTxDspFeatures* field of the RADIODEVCAPS structure.

Parameters

dwParam

Specifies the sampling rate, bits per sample and number of audio channels. The data is always in PCM format, the left channel before right channel when recording in stereo.

The parameters are specified by combining three RADIODSP_xxx flags, one for the sampling rate, one for the bits per sample and one for the number of channels (only those specified in *dwRxDspFeatures* or *dwTxDspFeatures* in the RADIODEVCAPS structure can be used). If any extra flags are set or if any flags are missing, the command will fail.

For example:

RADIODSP_11KHZ + RADIODSP_8BIT + RADIODSP_MONO will initiate an 11.025 kHz, 8 bit, single channel conversion.

Digitised data will automatically be sent to a plug-in with the PNR/T_DSPINBUFFULL notification (there is no need to call the PMR/T_DSPADDINBUF command).

Call PMR/T_DSPCLOSE when finished, passing the return value from this command in *dwParam*.

lpData

Not used.

Return Value

A unique 'DSP handle' if recording was successfully started, otherwise PLUGIN_CB_FAIL is returned (0x80000000).

PMR/T_DSPADDINBUF

The PMR_DSPADDINBUF and PMT_DSPADDINBUF commands are sent to the application to receive data from the DSP in a custom DSP application. This only works after a successful call to PMR/T_DSPSTART.

Parameters

dwParam

The 'DSP handle' that was returned from PMR/T_DSPOPEN.

cbData

Specifies the amount of data the plug-in wishes to receive from the DSP. When the buffer is full (or there is no more data from the DSP), the application issues a PNR/T_DSPINBUFFULL notification.

lpData

NULL.

Return Value

If successful, returns a unique 'Buffer ID', otherwise PLUGIN_CB_FAIL is returned (0x80000000).

PMR/T_DSPCLOSE

The PMR_DSPCLOSE and PMT_DSPCLOSE commands are sent to the application to close an active DSP process.

Parameters

dwParam

A 'DSP handle' that was returned from PMR/T_DSPSTART, PMR/T_DSPDACSTART or PMR/T_DSPADCSTART.

lpData

Not used.

Return Value

Zero if the command was successful, otherwise PLUGIN_CB_FAIL is returned (0x80000000).

PMR/T_DSPDACSTART

The PMR_DSPDACSTART and PMT_DSPDACSTART commands are sent to the application to initiate 'digital-analog conversion' of digitised audio data. A typical application is to enable audio playback from a hard disk that was recorded at an earlier time.

This function can only be used if the RADIODSP_DAC flag is set in the *dwRxDspFeatures* or *dwTxDspFeatures* field of the RADIODEVCAPS structure respectively.

Parameters

dwParam

Specifies the sampling rate, bits per sample and number of audio channels. The data is always in PCM format, the left channel before right channel when playback is in stereo.

The parameters are specified by combining three RADIODSP_xxx flags, one for the sampling rate, one for the bits per sample and one for the number of channels (only those specified in *dwDspFeatures* can be used). If any extra flags are set or if any flags are missing, the command will fail.

For example:

RADIODSP_11KHZ + RADIODSP_8BIT + RADIODSP_MONO will initiate an 11.025 kHz, 8 bit, single channel conversion.

To send the data to the DAC/DSP, call the PMR/T_DSPSENDBUF command.

Call PMR/T_DSPCLOSE when finished.

lpData

Not used.

Return Value

A unique 'DSP handle' if conversion was successfully started, otherwise PLUGIN_CB_FAIL is returned (0x80000000).

PMR/T_DSPINPUT

The PMR_DSPINPUT and PMT_DSPINPUT commands are sent to the application to select the DSP/ADC's source line.

Parameters

dwParam

Specifies the input number from 0 (the receiver's demodulator output or transmitter's audio input) to *iNumRxDspInputs*-1 or *iNumTxDspInputs*-1 specified in the RADIODEVCAPS structure.

lpData

Not used.

Return Value

Zero if the input was successfully selected, otherwise PLUGIN_CB_FAIL is returned (0x80000000).

See Also

PNR/T_DSPINPUT

PMR/T_DSPREADBYTE

The PMR_DSPREADBYTE and PMT_DSPREADBYTE commands are sent to the application to read one or more bytes from the DSP in a custom DSP application. This only works after a successful call to PMR/T_DSPSTART.

Parameters

dwParam

A 'DSP handle' that was returned from PMR/T_DSPSTART.

lpData

NULL if only one byte is to be read, otherwise, points to a buffer where the application will attempt to read *cbData* bytes from the DSP.

Return Value

If successful, the command returns the byte read from the DSP if *lpData* is NULL or returns the number of bytes read from the DSP if *lpData* is not NULL. If unsuccessful, `PLUGIN_CB_FAIL` is returned.

PMR/T_DSPSENDBUF

The `PMR_DSPSENDBUF` and `PMT_DSPSENDBUF` commands are sent to the application to send a block of data to the DSP for processing.

If `PMR/T_DSPSTART` or `PMR/T_DSPDACSTART` has not been successfully called, this function will fail.

Parameters

dwParam

A 'DSP handle' that was returned from `PMR/T_DSPSTART` or `PMR/T_DSPDACSTART`.

lpData

Pointer to a buffer of data to send to the DSP.

Return Value

A 'Buffer ID' if successful (this will be returned in the `PNR/T_DSPSENDBUFDONE` notification to inform the plug-in when the data has been sent) , otherwise `PLUGIN_CB_FAIL` is returned (0x80000000).

PMR/T_DSPSENDBYTE

The `PMR_DSPSENDBYTE` and `PMT_DSPSENDBYTE` commands are sent to the application to send one or more bytes to the DSP for processing. This only works after a successful call to `PMR/T_DSPSTART`.

Parameters

dwParam

A 'DSP handle' that was returned from `PMR/T_DSPSTART`.

lpData

Points to a buffer that contains several bytes to the DSP, the amount specified by *cbData*.

Return Value

If successful, the command returns the number of bytes sent, otherwise `PLUGIN_CB_FAIL` is returned.

PMR/T_DSPSTART

The `PMR_DSPSTART` and `PMT_DSPSTART` commands are sent to the application so the plug-in can initiate a custom DSP application. As this command is DSP dependant, the plug-in should first check the *szRx/TxDspManufacturer* and *szRx/TxDspProduct* fields to make sure that the specified DSP hardware is supported.

This function can only be used if the `RADIODSP_DSP` flag is set in the *dwRx/TxDspFeatures* field of the `RADIODEVCAPS` structure.

Call `PMR/T_DSPCLOSE` when finished.

Parameters

dwParam

Not used.

lpData

Points to a buffer than contains code to load into the DSP. This is DSP specific and is not translated in any way.

Return Value

A unique 'DSP handle' if the DSP code was started successfully, otherwise `PLUGIN_CB_FAIL` is returned (0x80000000).

PMR/T_EXTOSC

The `PMR_EXTOSC` and `PMT_EXTOSC` commands are sent to the application to control the selection of an external reference oscillator.

Support for the selectable inputs are specified by the presence of the `RADIOCAPS_EXTREFOSC` flag in the *dwRxFeatures* and *dwTxFeatures* field in the `RADIODEVCAPS` structure.

Parameters

dwParam

If zero is specified, the internal reference oscillator is used. If it is non-zero, an external reference oscillator is used.

lpData

Not used.

Return Value

Zero if the oscillator was successfully selected, otherwise `PLUGIN_CB_FAIL` is returned (0x80000000).

See Also

`PNR/T_EXTOSC`

PMR/T_FREQ

The `PMR_FREQ` and `PMT_FREQ` commands are sent to the application to change the frequency of a receiver or transmitter without changing the displayed frequency. This can be useful when the receiver or transmitter is used with ancillary frequency conversion hardware.

Parameters

dwParam

Specifies the frequency to tune to receiver or transmitter to. The first 31 bits is used to specify the frequency in Hz from 0 to 2.147 GHz. If bit 31 is set (MSB), the value in the first 31 bits is multiplied by ten, allowing the tuneable frequency range to be from 0 to 21.47 GHz with a minimum resolution of 10 Hz.

The frequency must lie within one of the receiver frequency ranges specified in the `RADIODEVCAPS` structure (see the *iNumRxFreqRanges* and *iRxFreqRangeOffset* fields for getting supported receiver bands and *iNumTxFreqRanges* and *iTxFreqRangeOffset* fields for supported transmitter bands).

lpData

Not used.

Return Value

Zero if the frequency was successfully set, otherwise `PLUGIN_CB_FAIL` is returned (0x80000000).

PMR/T_FREQUENCY

The PMR_FREQUENCY and PMT_FREQUENCY commands are sent to the application to change the frequency the receiver or transmitter is tuned to respectively. This also updates the appropriate frequency display.

Parameters

dwParam

Specifies the frequency to tune to receiver or transmitter to. The first 31 bits is used to specify the frequency in Hz from 0 to 2.147 GHz. If bit 31 is set (MSB), the value in the first 31 bits is multiplied by ten, allowing the tuneable frequency range to be from 0 to 21.47 GHz with a minimum resolution of 10 Hz.

The frequency must lie within one of the receiver frequency ranges specified in the RADIODEVCAPS structure (see the *iNumRxFreqRanges* and *iRxFreqRangeOffset* fields for getting supported receiver bands and *iNumTxFreqRanges* and *iTxFreqRangeOffset* fields for supported transmitter bands).

lpData

Not used.

Return Value

Zero if the frequency was successfully set, otherwise PLUGIN_CB_FAIL is returned (0x80000000).

See Also

PNR/T_FREQUENCY

PMR_AFC

The PMR_AFC command is sent to the application to control the receiver's AFC (automatic frequency control), which is used to lock the receiver to signals whose frequency is unknown or drifting. This feature is normally only available in FM, FSK and AM-SYNC modes. However in principle, it can be extended to any mode, given suitable hardware and/or software.

This command is only supported if the RADIORXCAPS_AFC flag is specified in the *dwRxFeatures* field of the RADIODEVCAPS structure.

Parameters

dwParam

If zero is specified, AFC is deactivated. If it is non-zero, AFC is activated.

lpData

Not used.

Return Value

Zero if the AFC was successfully set, otherwise PLUGIN_CB_FAIL is returned (0x80000000).

See Also

PNR_AFC

PMR_AGC

The PMR_AGC command is sent to the application to control the receiver's AGC (automatic gain control).

Support for switchable AGC (whether it can be switched off, on and several predefined speeds) is specified by the presence of the RADIORXCAPS_AGC flag in the *dwRxFeatures* field in the RADIODEVCAPS structure.

Support for adjustable AGC time-constants (ie. separate attack, hold and decay adjustments) is specified by the presence of the RADIORXCAPS_ADJAGC flag.

Parameters

dwParam

If this is zero (RXAGC_OFF), the AGC is deactivated.

If the value is positive, it specifies the overall AGC speed:

```
RXAGC_MEDIUM
RXAGC_SLOW
RXAGC_FAST
RXAGC_VSLOW
RXAGC_VFAST
```

Supported AGC speeds are specified by the *iAgcSpeeds* field of the RADIODEVCAPS structure.

If the value is negative, *lpData* points to an AGCEXPARAMS structure where the value specified for each member is defined as follows:

-1: Each of the 3 fields specifies an RXAGC_xxx constant as defined above.

-2: Each of the 3 fields specifies a time for the attack, hold and decay portions of the AGC in milliseconds. The range for the attack time is specifies by the *iMinAgcAttack* and *iMaxAgcAttack* fields of the AGCEXCAPS structure. *iMinAgcHold* and *iMaxAgcHold* specify the range for the hold time. *iMinAgcDecay* and *iMaxAgcDecay* specify the range for the decay time.

lpData

Points to an AGCEXPARAMS structure if *dwParam* is negative (each part of the AGC envelope is specified). Otherwise, this parameter is NULL.

Return Value

Zero if the AGC was successfully set, otherwise PLUGIN_CB_FAIL is returned (0x80000000).

See Also

PNR_AGC

PMR_ATTEN

The PMR_ATTEN command is sent to the application to control the setting of the receiver's RF input attenuator.

Support for the attenuator is specified by the presence of the RADIORXCAPS_ATTEN flag in the *dwRxFeatures* field in the RADIODEVCAPS structure.

Parameters

dwParam

Specifies the attenuation level from 0 (no attenuation) to the value specified by the *iMaxAtten* field in the RADIODEVCAPS structure. If the RADIOCAL_ATTEN flag is set in the *dwCalibrated* field, this value is specified in dB.

Some receivers support discrete attenuation levels (typically multiples of 5 or 6 dB) rather than continuously adjustable levels. The *iAttenStep* field in the RADIODEVCAPS structure specifies this granularity.

If a receiver has only an on/off attenuator, the *iMaxAtten* and *iAttenStep* values will be the same. To switch on the attenuator, this parameter must be *iMaxAtten*.

lpData

Not used.

Return Value

Zero if the attenuator was successfully set, otherwise PLUGIN_CB_FAIL is returned (0x80000000).

See Also

PNR_ATTEN

PMR_AUDIOSRC

The PMR_AUDIOSRC command is sent to the application select the audio source which is applied to the receiver's audio output amplifier.

Parameters

dwParam

Specifies the audio source number. The *iRxAudioSources* field in the RADIODEVCAPS structure defines the audio sources available for selection, where each bit set represents a supported source.

The defined sources include:

RXAUDIOSRC_RADIO - receiver demodulator
RXAUDIOSRC_EXT - external (line in)
RXAUDIOSRC_DSP - DSP/DAC

Setting the audio source to RXAUDIOSRC_DSP only works after a successful call to PMR_DSPADCSTART or PMR_DSPSTART.

lpData

Not used.

Return Value

Zero if the audio output was successfully selected, otherwise PLUGIN_CB_FAIL is returned (0x80000000).

See Also

PNR_AUDIOSRC

PMR_BALANCE

The PMR_BALANCE command is sent to the application to control the receiver's audio balance control, if the receiver supports stereo.

The support for this function is specified by the presence of the RADIORXCAPS_BALANCE flag in the *dwRxFeatures* field of the RADIODEVCAPS structure.

Parameters

dwParam

Zero if the balance is centred, positive if the balance is towards the right and negative if towards the left. The *iBalanceRange* field in the RADIODEVCAPS structure specifies the maximum range of this parameter.

If the RADIOCAL_BALANCE flag is set in the *dwCalibrated* field of the RADIODEVCAPS structure, this value is specified as a multiple of 0.1 dB.

lpData

Not used.

Return Value

Zero if the balance was successfully set, otherwise `PLUGIN_CB_FAIL` is returned (0x80000000).

See Also

`PNR_BALANCE`

PMR_BANDWIDTH

The `PMR_BANDWIDTH` command is sent to the application to control the receiver's IF bandwidth.

Parameters

dwParam

Specifies the IF bandwidth in Hz. The bandwidth range and controllability are defined in the `DEMODDEF` array in the `RADIODEVCAPS` structure, and therefore the bandwidth is dependent on the mode that is set.

If the *dwMinIfBw* field of the associated mode is –1, then only the *dwMaxIfBw* value can be set (if there is more than one definition for a single mode, any of the defined *dwMaxIfBw* values for the mode can be set). If *dwMaxIfBw* is zero, the bandwidth cannot be set or selected.

If the *dwMinIfBw* field is not zero, any bandwidth value can be set between this minimum and the maximum defined by *dwMaxIfBw* with a resolution of *dwIfBwStep*.

lpData

Not used.

Return Value

Zero if the IF bandwidth was successfully set, otherwise `PLUGIN_CB_FAIL` is returned (0x80000000).

See Also

`PNR_BANDWIDTH`

PMR_BLOCKSCAN

The `PMR_BLOCKSCAN` command is sent to the application for it to perform a scan of a specified range of frequencies, and either record the signal strengths for each frequency step, and/or stop when if finds a signal which equals or exceeds a predefined signal strength.

Parameters

dwParam

The low word specifies the scan rate in steps per second.

The high word specifies the squelch level, a value of -1 will make the application scan all frequencies regardless of the signal strength.

The *iMaxScanRate* field of the `DEMODDEF` for the current mode defines the maximum scan rate.

lpData

Points to an array of 32-bit integers where each entry specifies a frequency to scan. When the function is finished (by a `PNR_SCANFINISHED` notification), the signal strengths will be returned in the notification's *lpData* parameter.

Return Value

Zero if block scan was successfully started, otherwise PLUGIN_CB_FAIL is returned (0x80000000).

See Also

PMR_STOPSCAN

PMR_DEMODSIGNAL

The PMR_DEMODSIGNAL command can be sent only by demodulator plug-ins. It sends a buffer of samples from a specific point in the demodulator for other plug-ins that might need it. Samples can be modified to implement extra signal processings like audio signal conditioning.

Parameters

dwParam

A constant specifying the demodulator point where the samples have been obtained.

DEMODSIGNAL_IF	-	IF input
DEMODSIGNAL_IQ	-	I and Q samples before filtering
DEMODSIGNAL_IQ_FILTERED	-	I and Q samples after filtering
DEMODSIGNAL_AUDIO	-	audio output

cbData

The amount of memory occupied by the structure containing the samples.

lpData

Pointer to the structure containing the samples, DEMODSIGNALDATA.

PMR_IFGAIN

The PMR_IFGAIN command is sent to the application to control the receiver's IF gain.

Support for adjustable IF gain is specified by the presence of the RADIORXCAPS_IFGAIN flag in the *dwFeatures* field in the RADIODEVCAPS structure.

Parameters

dwParam

Specifies the IF gain level. This value can be negative (attenuated) or positive (amplified), with the limits specified by the *iMinIfGain* and *iMaxIfGain* values in the RADIODEVCAPS structure.

If the RADIOCAL_IFGAIN flag is set in the *dwCalibrated* field of the RADIODEVCAPS structure, the gain is specified in dB.

If the RADIORXCAPS_AGCMAXGAIN is set and the AGC is active, the value limits the maximum gain which can be achieved by AGC action.

lpData

Not used.

Return Value

Zero if the IF gain was successfully set, otherwise PLUGIN_CB_FAIL is returned (0x80000000).

See Also

PNR_IFGAIN

PMR_IFSHIFT

The PMR_IFSHIFT command is sent to the application to control the receiver's IF shift according to the current mode (or other mode if a mode is specified).

Parameters

dwParam

Specifies the IF shift in Hz. The available range of IF shift depends on the mode and the associated *dwMaxIfShift* field in the DEMODDEF structure. If the mode doesn't support IF shift, the *dwMaxIfShift* is set to zero.

lpData

If this is not NULL, then it points to a DWORD value that specifies the mode the IF shift is to be applied to. The mode is not set, but the IF shift level is stored in the application, and will be set when that mode is selected. *cbData* is set to four (or **sizeof(DWORD)**) if this is used.

Return Value

Zero if the IF shift was successfully set, otherwise PLUGIN_CB_FAIL is returned (0x80000000).

See Also

PNR_IFSHIFT

PMR_IFSPECTRUM

The PMR_IFSPECTRUM command is sent by a digital demodulator plug-in to the application to provide the spectrum resulted from the IF input signal. When receiving this command, the application, apart from using it, must send PNR_IFSPECTRUM notifications to all plug-ins.

Parameters

dwParam

Not used

cbData

The amount of memory occupied by the IF spectrum samples.

lpData

Pointer to the vector of IF spectrum samples. Each sample is stored using 32-bit unsigned integers with (2^32-1) corresponding to the maximum possible level.

PMR_LOUD

The PMR_LOUD command is sent to the application to select or deselect the receiver's audio output loudness compensation features, if available.

Support for loudness compensation is specified by the presence of the RADIORXCAPS_LOUD flag in the *dwFeatures* field in the RADIODEVCAPS structure.

Parameters

dwParam

Zero if loudness compensation is off, non-zero if it is on.

Loudness compensation usually boosts bass and treble frequencies at low volume levels, with the amount of boost reducing as the volume is increased.

lpData

Not used.

Return Value

Zero if loudness compensation was successfully set, otherwise PLUGIN_CB_FAIL is returned (0x80000000).

See Also

PNR_LOUD

PMR_MODE

The PMR_MODE command is sent to the application to change the receiver's demodulation mode.

Parameters

dwParam

The mode to set the receiver to:

RADIOMODE_CW	-	Continuous Wave
RADIOMODE_LSB	-	Lower Side Band
RADIOMODE_USB	-	Upper Side Band
RADIOMODE_AM	-	Amplitude Modulation (non-synchronous)
RADIOMODE_SAM	-	Amplitude Modulation (synchronous)
RADIOMODE_FMN	-	Frequency Modulation, Narrow (typ. 0 - 25 kHz)
RADIOMODE_FMM	-	Frequency Modulation, Medium (typ. 25 - 100 kHz)
RADIOMODE_FMW	-	Frequency Modulation, Wide (typ. > 100 kHz)
RADIOMODE_FSK	-	Frequency Shift Keying
RADIOMODE_DAB	-	Digital Audio Broadcasting
RADIOMODE_FM3	-	Frequency modulation with 3 kHz deviation
RADIOMODE_FM6	-	Frequency modulation with 6 kHz deviation
RADIOMODE_AMN	-	Narrow bandwidth amplitude modulation
RADIOMODE_ISB	-	Double side band amplitude modulation with supressed carrier
RADIOMODE_DSB	-	Independent side band amplitude modulation with supressed carrier
RADIOMODE_RDS	-	FMW with RDS sub-carrier decoding
RADIOMODE_MBS	-	FMW with MBS sub-carrier decoding
RADIOMODE_RBDS	-	FMW with RBDS sub-carrier decoding

The modes that the device supports are specified by the DEMODDEF array that is defined by the *iNumRxModes* and *iRxModeListOffset* fields in the RADIODEVCAPS structure.

Note that the IF bandwidth is specified by a separate command (PMR_BANDWIDTH). The provision of three different FM modes, according to bandwidth, is for convenience, as separate demodulators for these modes will often be used. However in these cases, it may still necessary to send the PMR_BANDWIDTH command to ensure that the correct IF filter is selected.

When one of the RDS, MBS or RBDS FMW modes are selected, any decoded data will be sent to the plug-in in a PNR_FMWDATA notification.

lpData

Not used.

Return Value

Zero if the mode was successfully set, otherwise PLUGIN_CB_FAIL is returned (0x80000000).

See Also

PNR_MODE

PNR_FMWDATA

PMR_MODEXDATA

The PMR_MODEXDATA command is sent to the application to control any supported mode-dependent extended features.

Parameters

dwParam

Depends on the mode the device is set to:

RADIOMODE_CW - BFO offset (up +/- *dwMaxExData*)

RADIOMODE_FMN,
RADIOMODE_FMM,
RADIOMODE_FMW - Base bandwidth

RADIOMODE_DAB - Digital audio broadcasting standard

others are reserved or not used.

For each supported extended setting, the *dwMaxExData* defines the maximum value(s) or range in the associated DEMODDEF structure.

lpData

Not used.

Return Value

Zero if the extended settings were successfully set, otherwise PLUGIN_CB_FAIL is returned.

See Also

PNR_MODEXDATA

PMR_MONO

The PMR_MONO command is sent to the application to force mono reception, when the hardware supports automatic mono/stereo switching.

Parameters

dwParam

Zero for automatic mono/stereo switching (or for default behaviour), non-zero to force mono.

lpData

Not used.

Return Value

Zero if the command was successful, otherwise PLUGIN_CB_FAIL is returned (0x80000000).

See Also

PNR_MONO

PMR_MUTE

The PMR_MUTE command is sent to the application to force muting of the receiver's audio output signal.

Parameters

dwParam

If this is zero, the output is not forcibly muted. If it is non-zero, the output is muted.

Note that this command does not override hardware muting which may be provided within the receiver for specific purposes, such as rendering inaudible any PLL lockup transients.

lpData

Not used.

Return Value

Zero if the mute was successfully set, otherwise PLUGIN_CB_FAIL is returned (0x80000000).

See Also

PNR_MUTE

PMR_NOISEBLANKER

The PMR_NOISEBLANKER command is sent to the application to control the operation of the receiver's internal noise blanker.

Support for this function is specified by the presence of the RADIORXCAPS_NOISEBLANKER flag in the *dwFeatures* field of the RADIODEVCAPS structure.

Parameters

dwParam

Zero if the noise blanker is to be deactivated, -1 if the noise blanker is to be set to auto-mode.

A positive number specifies the noise blanker threshold where the maximum threshold is defined by the *iMaxNbThreshold* field of the RADIODEVCAPS structure.

lpData

Not used.

Return Value

Zero if the noise blanker was successfully set, otherwise PLUGIN_CB_FAIL is returned (0x80000000).

See Also

PNR_NOISEBLANKER

PMR_NOISEREDUCT

The PMR_NOISEREDUCT command is sent to the application to control the operation of the receiver's noise reduction feature if it exists.

Support for this function is specified by the presence of the RADIORXCAPS_NOISEREDUCT flags in the *dwFeatures* field of the RADIODEVCAPS structure.

Parameters

dwParam

Zero if noise reduction is to be deactivated, otherwise it is a positive number where each value specifies a different type of noise reduction system. The *iMaxNoiseReduction* field of the RADIODEVCAPS structure defines the maximum value.

lpData

Not used.

Return Value

Zero if noise reduction command was successful, otherwise PLUGIN_CB_FAIL is returned (0x80000000).

See Also

PNR_NOISEREDUCT

PMR_NOTCH

The PMR_NOTCH command is sent to the application to control the operation of the receiver's notch filter. It controls both an automatic notch and a manual notch filter.

Whether an auto and/or manual notch is supported is by the presence of the RADIORXCAPS_AUTONOTCH and/or RADIORXCAPS_MANUALNOTCH flags in the *dwRxFeatures* field of the RADIODEVCAPS structure.

Parameters

dwParam

If set to zero, the notch filter is off. If set to –1, the automatic notch filter is enabled. If set to a positive number, the manual notch is enabled with the notch set at this value in Hz. The *iMaxNotchFreq* field of the RADIODEVCAPS structure defines the maximum frequency.

lpData

Not used.

Return Value

Zero if the notch filter was successfully set, otherwise PLUGIN_CB_FAIL is returned (0x80000000).

See Also

PNR_NOTCH

PMR_PREAMP

The PMR_PREAMP command is sent to the application to control the setting of the receiver's RF preamplifier.

Support for the preamplifier is specified by the presence of the RADIORXCAPS_PREAMP flag in the *dwFeatures* field in the RADIODEVCAPS structure.

Parameters

dwParam

Specifies the amplification level from 0 (no amplification) to the value specified by the *iMaxPreamp* field in the RADIODEVCAPS structure. If the RADIOCAL_ATTEN flag is set in the *dwCalibrated* field, this value is specified in dB.

Most receivers support discrete amplification levels (typically multiples 9 or 12 dB) rather than continuously adjustable levels. The *iPreampStep* field in the RADIODEVCAPS structure specifies this granularity.

If a receiver has only an on/off preamplifier, the *iMaxPreamp* and *iPreampStep* values will be the same (and to switch on the preamplifier, this parameter must be *iMaxPreamp*).

lpData

Not used.

Return Value

Zero if the preamplifier was successfully set, otherwise PLUGIN_CB_FAIL is returned (0x80000000).

See Also

PNR_PREAMP

PMR_RFINPUT

The PMR_RFINPUT command is sent to the application to select which antenna to use if the receiver has more than one selectable antenna connector.

Parameters

dwParam

Specifies the antenna input number from 0. The number of inputs if defined by the *iNumRfInputs* field in the RADIODEVCAPS structure.

On some receivers, each input receives a defined frequency range that may or may not cover the entire frequency range. If an input is selected that does not correspond correctly to the current receiver frequency, the quality of reception may be less than optimal. To obtain information on which inputs to use according to frequency, the FREQRANGE array defines each frequency range and which antenna input(s) the range can use.

lpData

Not used.

Return Value

Zero if the antenna input was successfully selected, otherwise PLUGIN_CB_FAIL is returned.

See Also

PNR_RFINPUT

PMR_SQUELCH

The PMR_SQUELCH command is sent to the application to set the receiver's squelch parameters.

Parameters

dwParam

Specifies an array of bits which indicate the type of squelch:

RXSQUELCH_SLEVEL	–	signal level
RXSQUELCH_NLEVEL	–	noise squelch
RXSQUELCH_CTCSS	–	CTCSS tone
RXSQUELCH_SYLLABIC	–	syllabic squelch
RXSQUELCH_DTMF	–	DTMF tone burst

RXSQUELCH_2TONE – 2-tone burst
RXSQUELCH_5TONE – 5-tone burst
RXSQUELCH_DPL – DPL

all other bits are reserved

The *iSquelchFeatures* field of the RADIODEVCAPS structure specifies supported squelch methods.

lpData

Points to a SQUELCHSETTINGS structure that contains the squelch parameters. Only those fields that correspond to the above set bits have to be set.

Return Value

Zero if the squelch parameters were successfully set, otherwise PLUGIN_CB_FAIL is returned.

See Also

PNR_SQUELCH

PNR_SQUELCHED

PMR_STOPSCAN

The PMR_STOPSCAN command is sent to the application to abort a block scan that has been started (with the PMR_BLOCKSCAN command).

Parameters

dwParam

Not used.

lpData

This can be NULL if the data is not required.

Alternatively, it can point to an array of 32-bit integers that will be filled by the application with all the signal levels for the frequencies that had been scanned since the most recent PMR_BLOCKSCAN command. The size of the array should be at least equal to the size specified in the call to PMR_BLOCKSCAN.

Return Value

The number of frequencies that had been scanned since the most recent PMR_BLOCKSCAN command, otherwise PLUGIN_CB_FAIL is returned (0x80000000).

PMR_TRACKID

The PMR_TRACKID command is sent to the application to control the receiver's trunk tracking capabilities. To use this feature, a PMR_TRUNKFREQ command must be issued to enable trunk decoding. Support for trunk tracking is specified by the presence of the RADIORXCAPS_TRUNKING flag in the *dwRxFeatures* field of the RADIODEVCAPS structure.

Parameters

dwParam

Specifies the radio ID to track, -1 if no tracking is to be performed.

lpData

Not used.

Return Value

Zero if the trunk tracking ID was successfully set, otherwise PLUGIN_CB_FAIL is returned.

PMR_TRUNKFREQ

The PMR_TRUNKFREQ command is sent to the application to control the receiver's trunk decoding and tracking capabilities. Support for trunk tracking is specified by the presence of the RADIORXCAPS_TRUNKING flag in the *dwRxFeatures* field of the RADIODEVCAPS structure.

To track a particular radio, use the PMR_TRACKID command.

Parameters

dwParam

Specifies the trunking system's control frequency in Hz. If zero is specified, the trunking feature is disabled.

lpData

Not used.

Return Value

Zero if the trunking frequency was successfully set, otherwise PLUGIN_CB_FAIL is returned.

See Also

PNR_TRUNKFREQ

PNR_TRUNKID

PMR_VOLUME

The PMR_VOLUME command is sent to the application to control the setting of the receiver output volume control.

Parameters

dwParam

Specifies the volume setting from zero to the maximum value, defined by the *iMaxVolume* field, in *iVolumeStep* steps.

If the RADIOCAL_VOLUME flag is set in the *dwCalibrated* field of the RADIODEVCAPS structure, this value is specified in dB.

lpData

Not used.

Return Value

Zero if the volume was successfully set, otherwise PLUGIN_CB_FAIL is returned (0x80000000).

See Also

PNR_VOLUME

PMT_ANTIVOX

The PMT_ANTIVOX command is sent to the application to change the transmitter's anti-vox gain.

Support for anti-vox adjustment is specified by the presence of the RADIOTXCAPS_ANTIVOX flag in the *dwTxFeatures* field of the RADIODEVCAPS structure.

Parameters

dwParam

Specifies the anti-vox gain from 0 to *iMaxAntiVox* specified in the RADIODEVCAPS structure.

If the RADIOCAL_ANTIVOX flag is set in the *dwCalibrated* field of the RADIODEVCAPS structure, this value is specified in dB.

lpData

Not used.

Return Value

Zero if the anti-vox gain was successfully set, otherwise PLUGIN_CB_FAIL is returned (0x80000000).

See Also

PNT_ANTIVOX

PMT_AUDIOPROC

The PMT_AUDIOPROC command is sent to the application to select the desired form of audio processing for the transmitter.

Parameters

dwParam

Currently, three bits are defined specifying which audio processing features are enabled:

TXAUDIOPROC_COMP - Compression
TXAUDIOPROC_CLIP - Clipping
TXAUDIOPROC_AGC - AGC

The *iAudioProcFlags* field of the RADIODEVCAPS structure specifies the audio processing features that are supported by the transmitter.

lpData

Points to a TXAUDIOPROC structure specifying the characteristics of each enabled processing feature.

Return Value

Zero if the audio processing features were successfully set, otherwise PLUGIN_CB_FAIL is returned.

See Also

PNT_AUDIOPROC

PMT_MODE

The PMT_MODE command is sent to the application to control the transmitter's modulation parameters.

Parameters

dwParam

The low-word contains the mode to set the transmitter to:

RADIOMODE_CW - Continuous Wave

RADIOMODE_LSB	-	Lower Side Band
RADIOMODE_USB	-	Upper Side Band
RADIOMODE_AM	-	Amplitude Modulation (non-synchronous)
RADIOMODE_FMN	-	Frequency Modulation, Narrow (typ. 0 - 25 kHz)
RADIOMODE_FMM	-	Frequency Modulation, Medium (typ. 25 - 100 kHz)
RADIOMODE_FMW	-	Frequency Modulation, Wide (typ. > 100 kHz)
RADIOMODE_FSK	-	Frequency Shift Keying
RADIOMODE_DAB	-	Digital Audio Broadcasting
RADIOMODE_FM3	-	Frequency modulation with 3 kHz deviation
RADIOMODE_FM6	-	Frequency modulation with 6 kHz deviation
RADIOMODE_AMN	-	Narrow bandwidth amplitude modulation
RADIOMODE_ISB	-	Double side band amplitude modulation with supressed carrier
RADIOMODE_DSB	-	Independent side band amplitude modulation with supressed carrier

The modes that the transmitter supports are specified by the MODDEF array that is defined by the *iNumTxModes* and *lpTxModeDefs* fields in the RADIODEVCAPS structure.

If the transmitter supports a secondary sub-carrier, the high-word contains the mode (as above) for the sub-carrier. If the high-word is zero, a sub-carrier is not transmitted.

lpData

Points to a MODPARAMS structure that defines where to find the mode dependant information.

Each mode has a different set of parameters. The following parameters are defined for each mode:

CW:	Not used.
LSB, USB:	*dw…Param1* specifies the 'peak envelope power'. The *dwMaxParam1* field in the MODDEF structure specifies the maximum limit. If the RADIOCAL_SSBMODPEP flag is set in the *dwCalibrated* field of the RADIODEVCAPS structure, this value is specified as a percentage of the max.
AM:	Pointer to a single DWORD that specifies the 'modulation depth'. The maximum limit is specifies by the *dwMaxParam1* field in the MODDEF structure. If the RADIOCAL_AMMODDEPTH flag is set in the *dwCalibrated* field of the RADIODEVCAPS structure, this value is specified as a percentage of the max.
FMN, FMM:	*dw…Param1* specifies the maximum frequency deviation either side of the carrier, and *dw…Param2* specifies the base bandwidth of the audio. If the RADIOCAL_FMDEV flag is set in the *dwCalibrated* field of the RADIODEVCAPS structure, the *dw…Param1* field is specified in Hz.
FMW:	*dw…Param1* and *dw…Param2* are the same as the other, while the rest of the fields are unique to FMW. The *dw…Param3* field specifies the frequency of the pilot tone for stereo transmissions (0 is specified for mono), the maximum defined by the *dwMaxParam3* field of the MODDEF structure. If the RADIOCAL_FMWPILOTTONE flag is specified in the *dwCalibrated* field, then this value is in Hz.
FSK:	The *dw…Param1* field specifies the lower frequency of the FSK transmission. The *dw…Param2* field specifies the frequency shift. The *dw…Param3* field specifies the baud rate and *dw…Param4* specifies the frequency shift 'shaping'.
DAB:	The *dw…Param1* field specifies the digital audio broadcasting standard: 0 = Eureka 147 1 = IBOC 2 = WordSpace 3 = DRM

The *dwSeconaryCarrierFreq* field specifies the sub-carrier frequency relative to the main transmitter frequency in Hz (use zero for the default sub-carrier offset for the mode).

Return Value

Zero if the power level was successfully set, otherwise PLUGIN_CB_FAIL is returned (0x80000000).

See Also

PNT_MODE

PMT_MODSRC

The PMT_MODSRC is sent to the application to connect the transmitter's modulator input to a different source.

Parameters

dwParam

Specifies the source number. The *iTxModSources* field in the RADIODEVCAPS structure defines the sources that can be selected, where each bit set represents a supported source.

The low-word specifies the primary modulation source and the high-word specifies the sub-carrier modulation source. The transmitter supports sub-carriers if the RADIOTXCAPS_SUBCARRIER flag is set in the *dwTxFeatures* field of the RADIODEVCAPS structure.

The defined sources include:

- TXMODSRC_MIC - Microphone
- TXMODSRC_EXT - External audio signal (from line-in connector)
- TXMODSRC_DSP - Signal supplied from computer via DAC and/or DSP
- TXMODSRC_KEY - Morse key
- TXMODSRC_MISC1 - Miscellaneous depending on transmitter (eg. internal digital modulator)
- TXMODSRC_MISC2 - Another miscellaneous input (eg. a dedicated circuit function)

Setting the modulation source to TXMODSRC_DSP only works after a successful call to PMR/T_DSPADCSTART or PMR/T_DSPSTART.

lpData

Not used.

Return Value

Zero if the source was successfully selected, otherwise PLUGIN_CB_FAIL is returned (0x80000000).

See Also

PNT_MODSRC

PMT_RFPOWER

The PMT_RFPOWER command is sent to the application to control the transmitter's peak output power.

Parameters

dwParam

Specifies the transmitter's peak output power from 0 (no power) to *iMaxTxPower*.

If the RADIOCAL_TXPOWER flag is set in the *dwCalibrated* field of the RADIODEVCAPS structure, this value is specified in mW (milliwatts).

lpData

Not used.

Return Value

Zero if the power level was successfully set, otherwise `PLUGIN_CB_FAIL` is returned (0x80000000).

See Also

`PNT_RFPOWER`

PMT_SELCALL

The `PMT_SELCALL` command is sent to the application to specify squelch and selective calling parameters of the transmission.

Parameters

dwParam

Specifies the selective calling type:

`TXSELCALL_NORMAL`	-	normal audio (no selective calling)
`TXSELCALL_CTCSS`	-	a CTCSS tone continuously superimposed on normal audio
`TXSELCALL_SINGLE`	-	a single tone burst followed by normal audio
`TXSELCALL_DTMF`	-	a DTMF burst followed by normal audio
`TXSELCALL_2TONE`	-	a two tone sequential burst followed by normal audio
`TXSELCALL_5TONE`	-	a five tone sequential burst followed by normal audio
`TXSELCALL_DPL`	-	audio with a DPL burst (digital private line)

The *iTxSelCallTypes* field in the `RADIODEVCAPS` structure defines the selective calling types that are supported.

lpData

Depends on *dwParam*:

TXSELCALL_NORMAL: not used.

TXSELCALL_CTCSS: points to a `SELCALL_CTCSS` structure:

dwToneFreq	Tone frequency in mHz (1000th's of Hz).
dwToneLevel	From 0 (silent) to *iMaxToneLevel*.

TXSELCALL_SINGLE: points to a `SELCALL_SINGLE` structure:

dwSotFreq	Tone frequency in Hz at start of transmission.
dwEotFreq	Tone frequency in Hz at end of transmission.
dwToneLevel	From 0 (silent) to *iMaxToneLevel*.
dwToneDuration	Duration of tone transmitted in milliseconds (up to *iMaxToneDuration*).

TXSELCALL_DTMF: points to a `SELCALL_DTMF` structure:

dwDtmfTone	DTMF tone pair number from 0 to 15.
dwToneLevel	From 0 (silent) to *iMaxToneLevel*.
dwToneDuration	Duration of tones transmitted in milliseconds (up to *iMaxToneDuration*).

TXSELCALL_2TONE: points to a `SELCALL_TWOTONE` structure:

dwToneFreq1	Specifies the initial tone frequency in Hz.
dwToneFreq2	Specifies the second tone frequency in Hz.

dwToneLevel From 0 (silent) to *iMaxToneLevel*.

dwReserved

TXSELCALL_5TONE: points to a `SELCALL_FIVETONE` structure:

dwToneFreqs[5] An array of five frequencies specifies the sequential five tones in Hz.

dwToneLevel From 0 (silent) to *iMaxToneLevel*.

dwReserved

TXSELCALL_DPL: points to a `SELCALL_DPL` structure:

dwReserved

Return Value

Zero if the squelch and selective calling parameters were successfully set, otherwise `PLUGIN_CB_FAIL` is returned.

See Also

`PNT_SELCALL`

PMT_TX

The `PMT_TX` command is sent to the application to activate or deactivate the transmitter.

Parameters

dwParam

Non-zero to enter transmit mode, zero to deactivate the transmitter.

lpData

Not used.

Return Value

Zero if the command was successful, otherwise `PLUGIN_CB_FAIL` is returned (0x80000000).

See Also

`PNT_TX`

PMT_XMTCTL

The `PMT_XMTCTL` command is sent to the application to select the transmitter's method of activation.

Parameters

dwParam

The low-word specifies a bit array where the specified combination of enabled initiators can activate the transmitter.

`TXINITIATE_MICSWITCH`	-	manual activation by microphone switch
`TXINITIATE_SECONDARY`	-	manual activation by secondary switch (eg. foot-switch)
`TXINITIATE_SOFTWARE`	-	manual activation by software (see `PMT_TX` command)
`TXINITIATE_VOX`	-	voice activated

The transmitter activates when one or more conditions exist.

The *iTxInitiators* field in the `RADIODEVCAPS` structure defines the initiators that are supported.

The high-word specifies the transmitter release delay from 0 (immediate release) to *iTxMaxReleaseDelay*. If the `RADIOCAL_TXRELEASE` flag is set in the *dwCalibrated* field, this value is in milliseconds.

lpData

Not used.

Return Value

Zero if the initiator was successfully set, otherwise `PLUGIN_CB_FAIL` is returned (0x80000000).

See Also

`PNT_XMTCTL`

XRS Notifications

If a notification passes a non-NULL pointer in the *lpData* parameter and the plug-in wishes to keep the data after the notification returns, the contents must be copied to local storage. Upon returning from the notification, any memory allocated for *lpData* is freed.

There are four classes of notifications: ones that apply only to receivers (`PNR_xxx`), ones that apply only to transmitters (`PNT_xxx`), global and those that apply to neither (`PN_xxx`).

PN_CAPABILITIES

The PN_CAPABILITIES message informs the plug-in that the capabilities of the receiver changed due to another plug-in starting/stopping. The plug-in must be able to handle the changes that affect it without restarting.

Parameters

dwParam

Not used.

cbData

The amount of memory occupied by the new RADIODEVCAPS structure.

lpData

Pointer to the new RADIODEVCAPS.

PN_CLOSE

The `PN_CLOSE` message notifies the plug-in that it ***must*** shutdown. This notification cannot be filtered out.

Parameters

dwParam

Zero if sent from the application (usually from user initiation) or non-zero if from another plug-in (from a `PM_STOPPLUGIN` command).

lpData

Not used.

See Also

`PM_CLOSED`

PN_DISABLED

The `PN_DISABLED` message notifies the plug-in when any discrete part of the application interface changes its status, where the parts can be enabled or disabled.

Parameters

dwParam

An array of flags where each flag set represents that feature disabled. See the `PM_DISABLE` command for more information.

lpData

Not used.

PN_MEMBANK

The PN_MEMBANK message notifies the plug-in when the active frequency memory bank has changed.

Parameters

dwParam

Specifies the active bank number from 0 to *iNumBanks*-1 specified in the RADIODEVCAPS structure.

lpData

Not used.

See Also

PM_SELECTBANK

PN_MEMCHANGE

The PN_MEMCHANGE message notifies the plug-in when a frequency memory record is modified.

Parameters

dwParam

Specifies the record number that was changed.

lpData

Points to a MEMORYENTRY structure that contains the settings for the modified frequency memory record.

See Also

PM_STOREMEM

PN_MEMFILE

The PN_MEMFILE message notifies the plug-in when a new frequency memory file is loaded.

Parameters

dwParam

Not used.

lpData

Points to a null-terminated string that specifies the loaded frequency memory file.

See Also

PM_SETMEMFILE

PM_GETMEMFILE

PN_MEMFOLDER

The PN_MEMFOLDER message notifies the plug-in when the active folder has changed.

Parameters

dwParam

Not used.

lpData

Points to a null-terminated string that specifies the active folder.

See Also

PM_OPENFOLDER

PN_MEMRECALL

The PN_MEMRECALL message notifies the plug-in when the device recalls settings from a frequency memory record.

Parameters

dwParam

Specifies the record number recalled from 0 or 1 to *dwMaxRecords* specified in the RADIODEVCAPS structure.

lpData

Not used.

See Also

PM_RECALLMEM

PN_MINIMIZED

The PN_MINIMIZED message notifies the plug-in when the device's window is minimised or restored.

Parameters

dwParam

Non-zero if the window is minimised, zero if it is restored (or maximised).

lpData

Not used.

See Also

PM_MINIMIZE

PN_PLUGINSTARTED

The PN_PLUGINSTARTED message notifies the plug-in when another plug-in is started.

Parameters

dwParam

Specifies the plug-in type (the value returned from xrsPluginInit).

lpData

Points to a null-terminated string that specifies the plug-in that was started.

See Also

PM_STARTPLUGIN

PN_PLUGINSTOPPED

The PN_PLUGINSTOPPED message notifies the plug-in when another plug-in has closed.

Parameters

dwParam

Specifies the plug-in type (the value returned from xrsPluginInit).

lpData

Points to a null-terminated string that specifies the plug-in that was stopped.

See Also

PM_STOPPLUGIN

PN_POWER

The PN_POWER message notifies the plug-in when the power state of the device changes.

Parameters

dwParam

Non-zero when the device is powered up, zero when the device is powered down.

lpData

Not used.

See Also

PM_POWER

PN_VISIBLE

The PN_VISIBLE message notifies the plug-in when the device's window is hidden or shown (usually under a plug-in's control).

Parameters

dwParam

Non-zero if the window is shown, zero if it is hidden.

lpData

Not used.

See Also

PM_VISIBLE

PNR/T_AUDIOFILTER

The PNR_AUDIOFILTER and PNT_AUDIOFILTER messages notifies the plug-in when audio filter settings are changed, and applies to both receivers and transmitters.

Parameters

dwParam

This can be one of the following filter types:

AUDIOFILTER_NONE	-	No filtering.
AUDIOFILTER_TONE	-	Bass, treble and optional mid-range tone filtering.
AUDIOFILTER_BANDPASS	-	Band-pass filtering.
AUDIOFILTER_PARAMETRIC	-	Parametric equaliser filtering.
AUDIOFILTER_GRAPHIC	-	Graphic equaliser filtering.

lpData

Depends on *dwParam*:

AUDIOFILTER_NONE:

Not used (will be NULL).

AUDIOFILTER_TONE:

Pointer to bass and treble DWORDs and a third mid-range level if supported. *cbSize* can equal eight or twelve depending on the presence of the mid-range value. If the RADIOCAL_TONE flag is specified in the *dwCalibrated* field, then these values are specified in dB.

AUDIOFILTER_BANDPASS:

Pointer to a low-pass and high-pass frequency in Hz. Both these values are DWORDs and therefore *cbSize* must be set to eight. The lowest and highest frequencies are specified by the *iMinBpFreq* and *iMaxBpFreq* fields of the RADIODEVCAPS structure.

AUDIOFILTER_PARAMETRIC:

Pointer to an array of PARAEQPARAMS structures where for each entry a centre frequency, Q and level parameters are specified.

AUDIOFILTER_GRAPHIC:

Pointer to an array of DWORDS where for each equaliser frequency specified in the GRAPHEQCAPS structure, a corresponding level is specified. The number of frequencies that are in the array is specified by the *iNumFreqs* field.

See Also

PMR/T_AUDIOFILTER

PNR/T_DSP

The PNR_DSP and PNT_DSP messages notifies the plug-in when the ADC, DAC and/or DSP activation status changes.

Parameters

dwParam

Bit:

RADIODSP_DAC	-	digital-analog conversion (from computer to DSP)
RADIODSP_ADC	-	analog-digital conversion (from DSP to computer)
RADIODSP_DSP	-	other DSP function (activated by another plug-in or in an application specific instance)

lpData

Not used.

See Also

PMR/T_DSPDACSTART

PMR/T_DSPADCSTART

PMR/T_DSPSTART

PNR/T_DSPINBUFFULL

The PNR_DSPINBUFFULL and PNT_DSPINBUFFULL messages notifies the plug-in when a requested read from the DSP has been completed. This can be issued from a PMR/T_DSPADCSTART or PMR/T_DSPADDINBUF command. This notification cannot be filtered out.

Parameters

dwParam

Specifies the 'Buffer ID' returned from PMR/T_DSPADDINBUF or zero if the message is from a PMR/T_DSPADCSTART command.

lpData

Points to a buffer that contains the data from the DSP.

PNR/T_DSPINPUT

The PNR_DSPINPUT and PNT_DSPINPUT messages notifies the plug-in when the DSP/ADC's input selection has changed.

Parameters

dwParam

Specifies the input number from 0 (the receiver's demodulator output or transmitter's input) to *iNumRx/TxDspInputs*-1 specified in the RADIODEVCAPS structure.

lpData

Not used.

See Also

PMR/T_DSPINPUT

PNR/T_DSPREQREAD

The PNR_DSPREQREAD and PNT_DSPREQREAD messages notifies the plug-in of a DSP generated read request (ie. the DSP has data to send to the plug-in). The plug-in should issue a PMR/T_DSPADDINBUF command in response. This notification cannot be filtered out.

Parameters

dwParam

Specifies the 'DSP handle' returned from PMR/T_DSPSTART.

cbData

Specifies the size of the buffer the DSP requested in bytes.

lpData

Not used.

PNR/T_DSPREQSEND

The PNR_DSPREQSEND and PNT_DSPREQSEND messages notifies the plug-in of a DSP generated send request (ie. the DSP wants to receive data). The plug-in should issue a PMR/T_DSPSENDBUF command in response. This notification cannot be filtered out.

Parameters

dwParam

Specifies the 'DSP handle' returned from PMR/T_DSPSTART.

cbData

Specifies the size of the buffer the DSP requested in bytes.

lpData

Not used.

PNR/T_DSPREQUEST

The PNR_DSPREQUEST and PNT_DSPREQUEST messages notifies the plug-in of a DSP generated request. The value is plug-in defined and the notification cannot be filtered out.

Parameters

dwParam

Specifies the 'DSP handle' returned from PMR/T_DSPSTART.

lpData

Points to a DWORD that contains the DSP notification code.

PNR/T_DSPSENDBUFDONE

The PNR_DSPSENDBUFDONE and PNT_DSPSENDBUFDONE messages notifies the plug-in when the PMR/T_DSPSENDBUF command has completed (ie. all the data in the buffer has been sent to the DSP). This notification cannot be filtered out.

Parameters

dwParam

Specifies the 'Buffer ID' returned from the PMR/T_DSPSENDBUF command.

lpData

Not used.

PNR/T_EXTOSC

The PNR_EXTOSC and PNT_EXTOSC messages notifies the plug-in when the reference oscillator source is switched.

Parameters

dwParam

Zero if the internal reference oscillator is used, one if the device is switched to an external reference oscillator.

lpData

Not used.

See Also

PMR/T_EXTOSC

PNR/T_FREQUENCY

The PNR_FREQUENCY and PNT_FREQUENCY messages notifies the plug-in when the receiver's or transmitter's frequency has changed.

Parameters

dwParam

Specifies the frequency the device is tuned to. The first 31 bits is used to specify the frequency (from 0 to 2.147 GHz). If bit 31 set (MSB), the value in the first 31 bits is multiplied by ten allowing the tuneable frequency range from 0 to 21.47 GHz (and minimum resolution of 10 Hz).

lpData

Not used.

See Also

PMR/T_FREQUENCY

PMR/T_FREQ

PNR_AFC

The PNR_AFC message notifies the plug-in when the receiver's AFC is activated or deactivated.

Parameters

dwParam

Zero if the AFC is deactivated, non-zero if it is activated.

lpData

Not used.

See Also

PMR_AFC

PNR_AGC

The PNR_AGC message notifies the plug-in when the receiver's AGC settings have changed.

Parameters

dwParam

If this is zero (RXAGC_OFF), the AGC is deactivated.

If the value is positive, it specifies the overall AGC speed:

```
RXAGC_MEDIUM
RXAGC_SLOW
RXAGC_FAST
RXAGC_VSLOW
RXAGC_VFAST
```

If the value is negative, *lpData* points to an AGCEXPARAMS structure where the value specified for each member is defined as follows:

-1: Each of the 3 fields specifies an RXAGC_xxx constant as defined above.

-2: Each of the 3 fields specifies a time for the attack, hold and decay portions of the AGC in milliseconds. The range for the attack time is specifies by the *iMinAgcAttack* and *iMaxAgcAttack* fields of the AGCEXCAPS structure. *iMinAgcHold* and *iMaxAgcHold* specify the range for the hold time. *iMinAgcDecay* and *iMaxAgcDecay* specify the range for the decay time.

lpData

Points to an AGCEXPARAMS structure if *dwParam* is negative (each part of the AGC envelope is specified). Otherwise, this parameter is not used.

See Also

PMR_AGC

PNR_ATTEN

The PNR_ATTEN message notifies the plug-in when the receiver's RF input attenuator status has changed.

Parameters

dwParam

Specifies the current RF input attenuation.

If the RADIOCAL_ATTEN flag is set in the *dwCalibrated* field of the RADIODEVCAPS structure, this value is specified in dB.

lpData

Not used.

See Also

PMR_ATTEN

PNR_AUDIOSRC

The PNR_AUDIOSRC message notifies the plug-in when the receiver's audio amplifier is switched to a different audio signal source.

Parameters

dwParam

Specifies the audio source number. The *iRxAudioSources* field in the RADIODEVCAPS structure defines the available audio sources, where each bit set represents a supported source.

The defined sources include:

RXAUDIOSRC_RADIO - receiver demodulator
RXAUDIOSRC_EXT - external (line in)
RXAUDIOSRC_DSP - DSP/DAC

lpData

Not used.

See Also

PMR_AUDIOSRC

PNR_BALANCE

The PNR_BALANCE message notifies the plug-in when the receiver's right/left audio balance setting changes.

Parameters

dwParam

Zero if the balance is centred, positive if the balance is towards the right and negative if towards the left. The *iBalanceRange* field in the RADIODEVCAPS structure specifies the maximum range of this parameter.

If the RADIOCAL_BALANCE flag is set in the *dwCalibrated* field of the RADIODEVCAPS structure, this value is specified as the difference in dB between the right and left channels.

lpData

Not used.

See Also

PMR_BALANCE

PNR_BANDWIDTH

The PNR_BANDWIDTH message notifies the plug-in when the receiver's IF bandwidth has changed.

Parameters

dwParam

Specifies the receiver's IF bandwidth in Hz for the current receiver mode. Each mode has independent IF bandwidth settings.

lpData

Not used.

See Also

PMR_BANDWIDTH

PNR_DEMODSIGNAL

The PN_DEMODSIGNAL dispatches buffers with samples from various points in digital demodulators either for study or supplementary signal processing.

Parameters

dwParam

A constant specifying the demodulator point where the samples have been obtained.

DEMODSIGNAL_IF	-	IF input
DEMODSIGNAL_IQ	-	I and Q samples before filtering
DEMODSIGNAL_IQ_FILTERED	-	I and Q samples after filtering
DEMODSIGNAL_AUDIO	-	audio output

cbData

The amount of memory occupied by the structure containing the samples.

lpData

Pointer to the structure containing the samples, DEMODSIGNALDATA.

PNR_FMWDATA

The PNR_FMWDATA notification sends decoded RDS/MBS/RBDS data from an FMW transmission to the plug-in. These notifications are only sent if the RADIOMODE_RDS, RADIOMODE_MBS or RADIOMODE_RBDS modes are selected.

This message cannot be filtered.

Parameters

dwParam

Not used.

lpData

Points to a buffer that contains the decoded data from the FMW transmission.

See Also

PMR_MODE

PNR_IFGAIN

The PNR_IFGAIN message notifies the plug-in when the manual IF gain level is changed.

Parameters

dwParam

Specifies the current IF gain level. This value can be either negative (attenuated) or positive (amplified), with the limits specified by the *iMinIfGain* and *iMaxIfGain* values in the RADIODEVCAPS structure.

If the RADIOCAL_IFGAIN flag is set in the *dwCalibrated* field of the RADIODEVCAPS structure, this value is specified in dB.

If the RADIORXCAPS_AGCMAXGAIN is set and the AGC is active, this value limits the maximum gain that can be achieved by AGC action.

lpData

Not used.

See Also

PMR_IFGAIN

PNR_IFSHIFT

The PNR_IFSHIFT message notifies the plug-in when the receiver's IF shift has changed. This only applies to the current receiver mode. The IF shift value can be different for each mode. This message is not sent when the current mode doesn't support IF shift (which can be determined from the receiver's DEMODDEF array in the RADIODEVCAPS structure).

Parameters

dwParam

Specifies the IF shift value in Hz.

lpData

Not used.

See Also

PMR_IFSHIFT

PNR_IFSPECTRUM

The PNR_IFSPECTRUM message notifies the plug-in that a digital demodulator plugin sent to the application the spectrum of its IF input signal. The plug-in must not affect the spectrum samples.

Parameters

dwParam

Not used

cbData

The amount of memory occupied by the IF spectrum samples.

lpData

Pointer to the vector of IF spectrum samples. Each sample is stored using 32-bit unsigned integers with (2^32-1) corresponding to the maximum possible level.

PNR_LOUD

The PNR_LOUD message notifies the plug-in when the receiver's loudness compensation status changes.

Parameters

dwParam

Zero if loudness compensation is off, non-zero if it is on.

Loudness compensation usually boosts bass and treble frequencies at low volume levels, with the amount of boost reducing as the volume is increased.

lpData

Not used.

See Also

PMR_LOUD

PNR_MODE

The PNR_MODE message notifies the plug-in when the receiver's mode has changed.

Parameters

dwParam

The mode the receiver is set to:

RADIOMODE_CW	-	Continuous Wave
RADIOMODE_LSB	-	Lower Side Band
RADIOMODE_USB	-	Upper Side Band
RADIOMODE_AM	-	Amplitude Modulation (non-synchronous)
RADIOMODE_SAM	-	Amplitude Modulation (synchronous)
RADIOMODE_FMN	-	Frequency Modulation, Narrow (typ. 0 - 25 kHz)

RADIOMODE_FMM - Frequency Modulation, Medium (typ .25 - 100 kHz)
RADIOMODE_FMW - Frequency Modulation, Wide (typ. > 100 kHz)
RADIOMODE_FSK - Frequency Shift Keying
RADIOMODE_DAB - Digital Audio Broadcasting (see PMT_MODE for supported standards)
RADIOMODE_FM3 - Frequency modulation with 3 kHz deviation
RADIOMODE_FM6 - Frequency modulation with 6 kHz deviation
RADIOMODE_AMN - Narrow bandwidth amplitude modulation
RADIOMODE_ISB - Double side band amplitude modulation with supressed carrier
RADIOMODE_DSB - Independent side band amplitude modulation with supressed carrier
RADIOMODE_RDS - FMW with RDS sub-carrier decoding
RADIOMODE_MBS - FMW with MBS sub-carrier decoding
RADIOMODE_RBDS - FMW with RBDS sub-carrier decoding

lpData

Not used.

See Also

PMR_MODE

PNR_MODEXDATA

The PNR_MODEXDATA message notifies the plug-in when the receiver's extended data for the current mode has changed.

Parameters

dwParam

Depends on the mode the device is set to:

RADIOMODE_CW - BFO offset (up +/- *dwMaxExData*)

RADIOMODE_FMN,
RADIOMODE_FMM,
RADIOMODE_FMW - Base bandwidth

RADIOMODE_DAB - DAB standard (see PMT_MODE for supported standards)

others are reserved or not used.

lpData

Not used.

See Also

PMR_MODEXDATA

PNR_MONO

The PNR_MONO message notifies the plug-in when the mono/stereo status of the receiver changes.

Parameters

dwParam

Zero if the received transmission is mono, 1 if it is a stereo transmission and -1 if the output is forced to mono (regardless of received transmission).

This message is only received if the receiver supports stereo reception (RADIORXCAPS_STEREO and/or RADIORXCAPS_FMWSTEREO).

lpData

Not used.

See Also

PMR_MONO

PNR_MUTE

The PNR_MUTE message notifies the plug-in when the receiver's mute status changes.

Parameters

dwParam

Zero if the audio output is not muted, non-zero if it is.

lpData

Not used.

See Also

PMR_MUTE

PNR_NOISEBLANKER

The PNR_NOISEBLANKER message notifies the plug-in when the receiver's noise blanker status has changed.

Parameters

dwParam

Zero if the noise blanker is deactivated, -1 if the noise blanker is set to auto-mode.

A positive number specifies the noise blanker threshold where the maximum threshold is defined by the *iMaxNbThreshold* field of the RADIODEVCAPS structure.

lpData

Not used.

See Also

PMR_NOISEBLANKER

PNR_NOISEREDUCT

The PNR_NOISEREDUCT message notifies the plug-in when the receiver's noise reduction filter status has changed.

Parameters

dwParam

Zero if noise reduction is deactivated, otherwise it is a positive number where each value specifies a different type of noise reduction system. The *iMaxNoiseReduction* field of the RADIODEVCAPS structure defines the maximum value.

lpData

Not used.

See Also

PMR_NOISEREDUCT

PNR_NOTCH

The PN_NOTCH message notifies the plug-in when the receiver's notch filter settings have changed.

Parameters

dwParam

Zero if the notch is turned off, -1 if the auto-notch is enabled or a positive value specifying the manual notch filter's frequency.

lpData

Not used.

See Also

PMR_NOTCH

PNR_PREAMP

The PNR_PREAMP message notifies the plug-in when the setting of the receiver's RF preamplifier has changed.

Parameters

dwParam

Specifies the receiver's RF preamplifier gain.

If the RADIOCAL_PREAMP flag is set in the *dwCalibrated* field of the RADIODEVCAPS structure, this value is specified in dB.

lpData

Not used.

See Also

PMR_PREAMP

PNR_RFINPUT

The PNR_RFINPUT message notifies the plug-in when the receiver's RF input selection has changed.

Parameters

dwParam

Specifies the current RF input number from 1 to *iNumRfInputs* defined in the RADIODEVCAPS structure. Specifying zero will select the first antenna that corresponds to the current frequency.

lpData

Not used.

See Also

PMR_RFINPUT

PNR_SCANFINISHED

The PNR_SCANFINISHED message notifies the plug-in that a PMR_BLOCKSCAN command that was issued has been completed. This notification cannot be filtered out.

Parameters

dwParam

Specifies the index of the frequency at which scanning stopped at. If all frequencies were scanned without stopping, then this will be –1.

lpData

Points to an array of DWORDs where each entry is the signal strength for the corresponding frequency passed in the PMR_BLOCKSCAN command.

PNR_SCANNER

The PNR_SCANNER message notifies the plug-in when the receiver's scanner status changes.

Parameters

dwParam

RADIOSCAN_IDLE - not scanning
RADIOSCAN_SCANNING - scanning
RADIOSCAN_PAUSED - paused

lpData

Not used.

PNR_SLEVEL

The PNR_SLEVEL message notifies the plug-in what the currently received signal strength is. Typically, this is called at regular intervals to keep the plug-in updated with the latest signal level (even if it has not changed).

Parameters

dwParam

The current received signal strength. This can be an arbitrary value from 0 to a maximum or in actual dBm. If the reading is in dBm, the RADIOCAL_SLEVEL flag is set in the *dwCalibrated* field of the RADIODEVCAPS structure.

cbData

The current received RAW signal stregth. This is 8-bit value obtained from DAC.

lpData

Not used.

PNR_SQUELCH

The PNR_SQUELCH message notifies the plug-in when the squelch settings have changed.

Parameters

dwParam

Specifies an array of bits that indicate what controls the squelch:

`RXSQUELCH_SLEVEL`	–	signal level
`RXSQUELCH_NLEVEL`	–	noise squelch
`RXSQUELCH_CTCSS`	–	CTCSS tone
`RXSQUELCH_SYLLABIC`	–	syllabic squelch
`RXSQUELCH_DTMF`	–	DTMF tone burst
`RXSQUELCH_2TONE`	–	2-tone burst
`RXSQUELCH_5TONE`	–	5-tone burst
`RXSQUELCH_DPL`	–	DPL

other bits are reserved

lpData

Pointer to `SQUELCHSETTINGS` structure. Only those fields that correspond to the above set bits are valid (the others are undefined).

See Also

`PMR_SQUELCH`

PNR_SQUELCHED

The `PNR_SQUELCHED` message notifies the plug-in when the squelch status changes.

Parameters

dwParam

Zero if the squelch is not active, non-zero if it is active.

lpData

Not used.

PNR_TRACKID

The `PNR_TRACKID` notification is sent to the plug-in when the receiver's trunk tracking status has changed.

Parameters

dwParam

Specifies the radio ID that is being tracked, -1 if no tracking is being performed.

lpData

Not used.

See Also

`PMR_TRACKID`

PNR_TRUNKFREQ

The `PNR_TRUNKFREQ` notification is sent to the plug-in when the receiver's trunk control frequency has changed.

Parameters

dwParam

Specifies the trunking system's control frequency in Hz. If zero is specified, the trunking feature is disabled.

lpData

Not used.

See Also

PMR_TRUNKFREQ

PNR_TRUNKID

The PNR_TRUNKID notification is sent to the plug-in when a trunk ID is decoded and no tracking is being performed.

Parameters

dwParam

Specifies the decoded trunking radio ID. This value can be used in the PMR_TRACKID command to track the radio.

lpData

Not used.

PNR_VOLUME

The PNR_VOLUME message notifies the plug-in when the receiver's volume level has changed.

Parameters

dwParam

Specifies the audio volume setting from 0 to *iMaxVolume* specified in the RADIODEVCAPS structure.

If the RADIOCAL_VOLUME flag is set in the *dwCalibrated* field of the RADIODEVCAPS structure, this value is specified in dB.

lpData

Not used.

See Also

PMR_VOLUME

PNT_ANTIVOX

The PNT_ANTIVOX message notifies the plug-in when the transmitter's anti-vox gain has changed.

Parameters

dwParam

Specifies the anti-vox gain from 0 to *iMaxAntiVox* specified in the RADIODEVCAPS structure.

If the RADIOCAL_ANTIVOX flag is set in the *dwCalibrated* field of the RADIODEVCAPS structure, this value is specified in dB.

lpData

Not used.

See Also

PMT_ANTIVOX

PNT_AUDIOPROC

The PNT_AUDIOPROC message notifies the plug-in when the audio input processing of the transmitter has changed.

Parameters

dwParam

Currently, three bits are defined specifying which audio processing features are enabled:

TXAUDIOPROC_COMP	-	Compression
TXAUDIOPROC_CLIP	-	Clipping
TXAUDIOPROC_AGC	-	AGC

lpData

This will point to a TXAUDIOPROC structure specifying the characteristics of each enabled processing feature.

See Also

PMT_AUDIOPROC

PNT_MEASUREMENT

The PNT_MEASUREMENT message notifies the plug-in of various measurements which may be performed on the transmitter.

Parameters

dwParam

0: VSWR in 0.01 increments

1: Forward & reverse voltage from a directional coupler.

2: Forward & reverse power in mW

3: Power amplifier collector current in mA

4: Heatsink temperature in 0.1°C increments

5: Power amplifier supply voltage in mV

6: ALC voltage in mV

lpData

Depends on *dwParam*:

0: A DWORD value that specifies the current VSWR (Voltage Standing Wave Ratio) to two decimal places (therefore the minimum value is 100 = 1.00).

1: Two DWORD values that specify the forward and reverse voltage. The 1st DWORD specifies the forward voltage and the 2nd DWORD specifies the reverse (reflected) voltage.

2: Two DWORD values that specify the forward and reverse power in mW. The 1st DWORD specifies the forward power and the 2nd DWORD specifies the reverse (reflected) power.

3: A DWORD value that specifies the amplifier's output transistor collector current in mA.

4: A DWORD value that specifies the amplifier's heatsink temperature in °C to one decimal place (therefore 562 represents 56.2°C).

5: A DWORD value that specifies the amplifier's supply voltage in mV.

6: A DWORD value that specifies the ALC (auto-level control) voltage from an external high-power amplifier in mV.

PNT_MODE

The PNT_MODE notification is sent to the plug-in when the transmitter's modulation parameters change.

Parameters

dwParam

The low-word contains the mode the transmitter is set to:

RADIOMODE_CW	-	Continuous Wave
RADIOMODE_LSB	-	Lower Side Band
RADIOMODE_USB	-	Upper Side Band
RADIOMODE_AM	-	Amplitude Modulation (non-synchronous)
RADIOMODE_FMN	-	Frequency Modulation, Narrow (typ. 0 - 25 kHz)
RADIOMODE_FMM	-	Frequency Modulation, Medium (typ. 25 - 100 kHz)
RADIOMODE_FMW	-	Frequency Modulation, Wide (typ. > 100 kHz)
RADIOMODE_FSK	-	Frequency Shift Keying
RADIOMODE_DAB	-	Digital Audio Broadcasting
RADIOMODE_FM3	-	Frequency modulation with 3 kHz deviation
RADIOMODE_FM6	-	Frequency modulation with 6 kHz deviation
RADIOMODE_AMN	-	Narrow bandwidth amplitude modulation
RADIOMODE_ISB	-	Double side band amplitude modulation with supressed carrier
RADIOMODE_DSB	-	Independent side band amplitude modulation with supressed carrier

The modes that the transmitter supports are specified by the MODDEF array that is defined by the *iNumTxModes* and *iTxModeListOffset* fields in the RADIODEVCAPS structure.

If the transmitter supports a secondary sub-carrier, the high-word contains the mode (as above) for the sub-carrier. If the high-word is zero, a sub-carrier is not transmitted.

lpData

Points to a MODPARAMS structure that defines where to find the mode dependant information.

Each mode has a different set of parameters. The following parameters are defined for each mode:

CW:	Not used.
LSB, USB:	*dw…Param1* specifies the 'peak envelope power'. The *dwMaxParam1* field in the MODDEF structure specifies the maximum limit. If the RADIOCAL_SSBMODPEP flag is set in the *dwCalibrated* field of the RADIODEVCAPS structure, this value is specified as a percentage of the max.
AM:	Pointer to a single DWORD that specifies the 'modulation depth'. The maximum limit is specifies by the *dwMaxParam1* field in the MODDEF structure. If the RADIOCAL_AMMODDEPTH flag is set in the *dwCalibrated* field of the RADIODEVCAPS structure, this value is specified as a percentage of the max.
FMN, FMM:	*dw…Param1* specifies the maximum frequency deviation either side of the carrier, and *dw…Param2* specifies the base bandwidth of the audio. If the RADIOCAL_FMDEV flag is set in the *dwCalibrated* field of the RADIODEVCAPS structure, the *dw…Param1* field is specified in Hz.

FMW: *dw…Param1* and *dw…Param2* are the same as the other, while the rest of the fields are unique to FMW.

The *dw…Param3* field specifies the frequency of the pilot tone for stereo transmissions (0 is specified for mono), the maximum defined by the *dwMaxParam3* field of the `MODDEF` structure. If the `RADIOCAL_FMWPILOTTONE` flag is specified in the *dwCalibrated* field, then this value is in Hz.

FSK: The *dw…Param1* field specifies the lower frequency of the FSK transmission. The *dw…Param2* field specifies the frequency shift. The *dw…Param3* field specifies the baud rate and *dw…Param4* specifies the frequency shift 'shaping'.

DAB: The *dw…Param1* field specifies the digital audio broadcasting standard:

0 = Eureka 147
1 = IBOC
2 = WordSpace
3 = DRM

The *dwSeconaryCarrierFreq* field specifies the sub-carrier frequency relative to the main transmitter frequency in Hz (use zero for the default sub-carrier offset for the mode).

See Also

`PMT_MODE`

PNT_MODSRC

The `PNT_MODSRC` notification is sent to the plug-in when the transmitter's modulator input is connected to a different source.

Parameters

dwParam

Specifies the source number. The *iTxModSources* field in the `RADIODEVCAPS` structure defines the sources that can be selected, where each bit set represents a supported source.

The low-word specifies the primary modulation source and the high-word specifies the sub-carrier modulation source. The receiver supports sub-carriers if the `RADIOTXCAPS_SUBCARRIER` flag is set in the *dwTxFeatures* field of the `RADIODEVCAPS` structure.

The defined sources include:

`TXMODSRC_MIC` - Microphone
`TXMODSRC_EXT` - External audio signal (from line-in connector)
`TXMODSRC_DSP` - Signal supplied from computer via DAC and/or DSP
`TXMODSRC_KEY` - Morse key
`TXMODSRC_MISC1` - Miscellaneous depending on transmitter (eg. internal digital modulator)
`TXMODSRC_MISC2` - Another miscellaneous input (eg. a dedicated circuit function)

lpData

Not used.

See Also

`PMT_MODSRC`

PNT_RFPOWER

The `PNT_RFPOWER` notification is sent to the plug-in when the transmitter's peak output power setting has changed.

Parameters

dwParam

Specifies the transmitter's peak output power from 0 (no power) to *iMaxTxPower*.

If the RADIOCAL_TXPOWER flag is set in the *dwCalibrated* field of the RADIODEVCAPS structure, this value is specified in mW (milliwatts).

lpData

Not used.

See Also

PMT_RFPOWER

PNT_SELCALL

The PNT_SELCALL notification is sent to the plug-in when the squelch and selective calling parameters of the transmission have changed.

Parameters

dwParam

Specifies the selective calling type:

TXSELCALL_NORMAL	-	normal audio (no selective calling)
TXSELCALL_CTCSS	-	a CTCSS tone continuously superimposed on normal audio
TXSELCALL_SINGLE	-	a single tone burst followed by normal audio
TXSELCALL_DTMF	-	a DTMF burst followed by normal audio
TXSELCALL_5TONE	-	a five tone sequential burst followed by normal audio
TXSELCALL_DPL	-	audio with a DPL burst (digital private line)

The *iTxSelCallTypes* field in the RADIODEVCAPS structure defines the selective calling types that are supported.

lpData

Depends on *dwParam*:

TXSELCALL_NORMAL: not used.

TXSELCALL_CTCSS: points to a SELCALL_CTCSS structure:

dwToneFreq	Tone frequency in mHz (1000th's of Hz).
dwToneLevel	From 0 (silent) to *iMaxToneLevel*.

TXSELCALL_SINGLE: points to a SELCALL_SINGLE structure:

dwSotFreq	Tone frequency in Hz at start of transmission.
dwEotFreq	Tone frequency in Hz at end of transmission.
dwToneLevel	From 0 (silent) to *iMaxToneLevel*.
dwToneDuration	Duration of tone transmitted in milliseconds (up to *iMaxToneDuration*).

TXSELCALL_DTMF: points to a SELCALL_DTMF structure:

dwDtmfTone	DTMF tone pair number from 0 to 15.
dwToneLevel	From 0 (silent) to *iMaxToneLevel*.
dwToneDuration	Duration of tones transmitted in milliseconds (up to *iMaxToneDuration*).

TXSELCALL_2TONE: points to a SELCALL_TWOTONE structure:

dwToneFreq1	Specifies the initial tone frequency in Hz.

dwToneFreq2 Specifies the second tone frequency in Hz.

dwToneLevel From 0 (silent) to *iMaxToneLevel*.

dwReserved

TXSELCALL_5TONE: points to a SELCALL_FIVETONE structure:

dwToneFreqs[5] An array of five frequencies specifies the sequential five tones in Hz.

dwToneLevel From 0 (silent) to *iMaxToneLevel*.

dwReserved

TXSELCALL_DPL: points to a SELCALL_DPL structure:

dwReserved

See Also

PMT_SELCALL

PNT_TX

The PNT_TX message notifies the plug-in when the transmitter is activated.

Parameters

dwParam

Non-zero when the device enters transmit mode, zero if the device's transmitter deactivates.

lpData

Not used.

See Also

PMT_TX

PNT_XMTCTL

The PNT_XMTCTL notification is sent to the plug-in when the transmitter's method of activation has changed.

Parameters

dwParam

The low-word specifies a bit array where the specified combination of enabled initiators can activate the transmitter.

TXINITIATE_MICSWITCH	-	manual activation by microphone switch
TXINITIATE_SECONDARY	-	manual activation by secondary switch (eg. foot-switch)
TXINITIATE_SOFTWARE	-	manual activation by software (see PMT_TX command)
TXINITIATE_VOX	-	voice activated

The transmitter activates when one or more conditions exist.

The *iTxInitiators* field in the RADIODEVCAPS structure defines the initiators that are supported.

The high-word specifies the transmitter release delay from 0 (immediate release) to *iTxMaxReleaseDelay*. If the RADIOCAL_TXRELEASE flag is set in the *dwCalibrated* field, this value is in milliseconds.

lpData

Not used.

See Also

PMT_XMTCTL