

## StreamXpress Remote Control API

- Remote Control of StreamXpress
- SOAP based
- C++ Library and WSDL file

### Features

- Remote control of *StreamXpress* functions on same PC or from other PC on the network
- Client and *StreamXpress* communicate using industry-standard SOAP messages
- WSDL file enables automatic generation of SOAP proxy
- Remote-Control (RC) license required on DekTec device that runs *StreamXpress*

Copyright © 2011-2021 by DekTec Digital Video B.V.

DekTec Digital Video B.V. reserves the right to change products or specifications without notice. Information furnished in this document is believed to be accurate and reliable, but DekTec Digital Video assumes no responsibility for any errors that may appear in this material.

## Table of Contents

<b>Features</b> .....	1	Struct SpRcRfPars	39
<b>Table of Contents</b> .....	2	Struct SpRcSpiPars	40
<b>SpRcApi – Revision History</b> .....	3	Struct SpRcTsoipPars	41
<b>1. Using SpRcApi</b> .....	4	SpRcClient::ClearErrors	45
1.1. Introduction	4	SpRcClient::GetAsiPars	46
1.2. Running StreamXpress	4	SpRcClient::GetChannelModellingPars	47
1.3. Client-Server Communication	4	SpRcClient::GetCmmbPars	48
1.4. Using SpRcApi	4	SpRcClient::GetDvbT2Group	49
<b>2. Creating a Proxy from WSDL</b> .....	5	SpRcClient::GetDvbT2Pars	50
2.1. SpRc.wsdl	5	SpRcClient::GetHwNoisePars	51
2.2. Generating the Proxy	5	SpRcClient::GetIsdbtPars	52
<b>3. Using the Static Link Library</b> .....	6	SpRcClient::GetLqGain	53
3.1. Including and Linking SpRcApi	6	SpRcClient::GetModPars	54
3.2. Checking return codes	6	SpRcClient::GetPayoutInfo	55
3.3. Connecting to StreamXpress	6	SpRcClient::GetPayoutStatus	56
3.4. Playing a file	6	SpRcClient::GetRfPars	57
<b>SpRcClient – Session Interface</b> .....	8	SpRcClient::GetSignalSource	58
SpRcClient::CloseSession	8	SpRcClient::GetSpiPars	59
SpRcClient::CreateSpRcClient	9	SpRcClient::GetTsgPars	60
SpRcClient::GetRemoteVersion	10	SpRcClient::GetTsoipPars	61
SpRcClient::GetRemoteDtapiVersion	11	SpRcClient::GetUseNit	62
SpRcClient::GetVersion	12	SpRcClient::Normalise	63
SpRcClient::OpenSession	13	SpRcClient::OpenChannelModellingFile	64
<b>SpRcClient – Application Common Interface</b> .....	14	SpRcClient::OpenFile	65
SpRcClient::GetAppInfo	14	SpRcClient::SaveChannelModellingSettings	66
SpRcClient::ShowWindow	15	SpRcClient::SaveSettings	67
<b>SpRcClient – Port Selection Interface</b> .....	16	SpRcClient::SetAsiPars	68
Struct PortDesc	16	SpRcClient::SetChannelModellingPars	69
SpRcClient::ScanPorts	19	SpRcClient::SetCmmbPars	70
SpRcClient::SelectPort	20	SpRcClient::SetDvbT2Group	71
SpRcClient::SelectDtaPlus	21	SpRcClient::SetDvbT2Pars	72
<b>SpRcClient – Payout Interface</b> .....	22	SpRcClient::SetHwNoisePars	73
Struct SpRcAsiPars	22	SpRcClient::SetIsdbtPars	74
Struct SpRcCmmbPars	24	SpRcClient::SetLqGain	75
Struct SpRcCmPars	25	SpRcClient::SetLoopFlags	76
Struct SpRcCmPaths	26	SpRcClient::SetModPars	77
Struct SpRcDvbT2Group	27	SpRcClient::SetPayoutState	78
Struct SpRcDvbT2Pars	28	SpRcClient::SetRfPars	79
Struct SpRcHwNoisePars	30	SpRcClient::SetSignalSource	80
Struct SpRclsdbtLayerPars	31	SpRcClient::SetSpiPars	81
Struct SpRclsdbtPars	33	SpRcClient::SetTsgPars	82
Struct SpRcModPars	35	SpRcClient::SetTsoipPars	83
Struct SpRcPayoutInfo	36	SpRcClient::SetTsRate	84
Struct SpRcPayoutStatus	38	SpRcClient::SetUseNit	85
		SpRcClient::WaitForCondition	86

## SpRcApi – Revision History

Version	Date	Change Description
v1.9.0.16	2021.03.16	<ul style="list-style-type: none"> <li>• Add support for VC16 (Visual Studio 2019) libraries</li> <li>• Add support for ATSC3.0 STLTP (Studio-to-Transmitter Link Transport Protocol)</li> <li>• Add support for Digital Radio Mondial (DRM/DRM+)</li> <li>• Add support for DVB-T2 version configuration</li> </ul>
v1.7.0.14	2019.08.05	<ul style="list-style-type: none"> <li>• Add support for VC11, V12, VC14 and VC15 libraries</li> <li>• Add support for ISDB-S3</li> </ul>
v1.6.0.13	2015.01.08	<ul style="list-style-type: none"> <li>• Add GefTsgPars()/SetTsgPars() to control test-generator options</li> </ul>
v1.5.0.12	2014.06.25	<ul style="list-style-type: none"> <li>• Extend SpRcTsoipPars to support double buffering on DTA-2162</li> <li>• Add GetIqGain()/SetIqGain() functions to allow changing IQ gain when playing out IQ signals</li> <li>• Add GetRemoteDtapiVersion() to check the DTAPI version used to build StreamXpress</li> </ul>
v1.4.3.10	2014.04.24	<ul style="list-style-type: none"> <li>• Add support for DVB-S2X and S2L3</li> </ul>
v1.4.2.9	2014.02.13	<ul style="list-style-type: none"> <li>• Bugfix release to fix linking problems</li> </ul>
v1.4.1.8	2013.03.01	<ul style="list-style-type: none"> <li>• Add support for DtaPlus device</li> <li>• Add SetRemux() function to control remultiplexing on modulator ports</li> </ul>
v1.3.0.6	2012.05.15	<ul style="list-style-type: none"> <li>• Add Normalise function</li> <li>• Add Open and SaveChannelModellingSettings function</li> <li>• Add Set and GetSignalSource functions</li> <li>• Add Set and GetUseNit functions</li> <li>• Add variables to RfPars struct: Speclnv, CW, RfEnabledOnStop</li> <li>• Add SaveSettings function</li> </ul>
v1.2.0.5	2011.09.27	<ul style="list-style-type: none"> <li>• Add Set and GetCmmbPars functions</li> <li>• Add Set and GetDvbT2Group functions</li> <li>• Add Set and GetHwNoisePars functions</li> <li>• WSDL namespaces, removed "http://localhost:80/SpRc.wsdl"</li> <li>• Add SpRcApiNET .NET WSDL proxy wrapper example code</li> <li>• Synchronized documentation with SpRc source code</li> </ul>
v1.1.0.4	2010.12.08	<ul style="list-style-type: none"> <li>• Add Set and GetChannelModellingPars functions</li> <li>• Release of WS-I Basic Profile 1.0a compliant WSDL description; used for automatic proxy generation (for example to be used by Tools like Visual Studio.NET and LabView)</li> <li>• Add ShowWindow function for hiding/showing StreamXpress Window</li> </ul>
v1.0.0.1	2009.02.16	<ul style="list-style-type: none"> <li>• First release to the field</li> </ul>

## 1. Using SpRcApi

### 1.1. Introduction

The StreamXpress Remote-Control API (**SpRcApi**) enables a client application to remotely control the *StreamXpress*, in order to automate the play out of streams. The client can be running on the same PC as the *StreamXpress* or on another PC in the network. Most functions available in the *StreamXpress* GUI are also available through the **SpRcApi**.

In this document, the remotely controlled *StreamXpress* will be interchangeably referred to as "server", "playout server" or "*StreamXpress*".

The client is the application that wishes to remotely control the *StreamXpress*.

### 1.2. Running StreamXpress

To run the *StreamXpress* as playout server it is required that:

1. The DekTec device used for playout with *StreamXpress* contains a remote control (RC) license;
2. The *StreamXpress* is started with the `-rc` option, followed by the TCP port number that is used to connect, e.g. `-rc 9000`.

### 1.3. Client-Server Communication

Client and server communicate with each other using SOAP calls over an IP network, or using the local host. SOAP is a standardized protocol that uses messages formatted in XML to execute remote procedure calls.

Currently, SpRcApi supports version 1.1 of the SOAP protocol.

### 1.4. Using SpRcApi

Basically there are 4 ways to deploy SpRcApi:

- Use the WSDL file in applications that can import WSDL files, e.g. Labview.
- Use the WSDL file to automatically generate client code, commonly called a "proxy". The proxy allows you to call the API methods. Section 2 describes an

example to create a proxy with Visual Studio

- Use the SpRcApiNET example code that includes a .NET "wrapper" for the WSDL generated proxy code.
- For clients written in C++, a library is available that makes the remote-control methods available as C++ methods. This is explained further in Section 3.

The main part of this document describes the classes and API calls in SpRcApi, in C++ syntax. The C++ descriptions are easy to map the equivalent SOAP calls because the WSDL API is structured in a similar way. There is one exception: the SpRcClient constants cannot be used directly, the equivalent integer numbers shall be used.

## 2. Creating a Proxy from WSDL

### 2.1. SpRc.wsdl

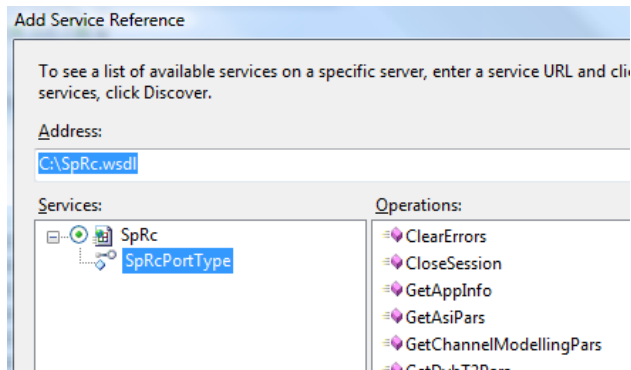
The WSDL is contained in the file SpRc.wsdl, which is available in SpRcApi.zip.

The SpRc.wsdl description is a WS-I Basic Profile 1.0a compliant web-service description. This file can be used in several tools to automatically generate client proxy code. For example, LabView and Visual Studio have the possibility to import the WSDL file.

### 2.2. Generating the Proxy

The steps below describe how to generate proxy code and how to use this code in a C# program in a Visual Studio environment:

1. After creating a C# project, right mouse click on the 'References' and choose option 'Add Service Reference...'
2. In the 'Add Service Reference' dialog select the SpRc.wsdl location, change the Namespace and click on the 'OK' button. This will start the proxy code generation.



Some tools don't accept a file location, for these tools you can use the following address to locate SpRc.wsdl: <http://www.dektec.com/Products/Apps/DT-C-300/SpRc.wsdl>

3. Adapt the generated 'app.config' file to the correct remote IP address and used port number.

```
<client>
  <endpoint
    address = "http://localhost:9000"
    binding = "basicHttpBinding"
    bindingConfiguration = "SpRc"
    contract = "SpRc.SpRcPortType"
    name = "SpRc" />
</client>
```

4. The following C# code shows how to instantiate the generated remote-control client proxy and how to do a few remote-control calls.

The Interface calls and structs are similar to the described SpRcClient API but some differences exist:

- a. The result code can be returned via an out parameter and a parameter can be returned via the return value.
- b. SpRcClient constants cannot be used; the equivalent integer numbers shall be used.

```
// Instantiate .NET generated RC client
SpRc.SpRcPortTypeClient c1 =
    new SpRc.SpRcPortTypeClient();

// Open a session
uint res = c1.OpenSession();

// Scan Ports
SpRc.PortDesc[] ports =
    c1.ScanPorts(out res);

// Select DVB-C Modulation
// ( 9 : SPRC_MOD_J83A )
res = c1.SelectPort(ports[0].Serial,
    ports[0].Port, 9, out status);
```

## 3. Using the Static Link Library

### 3.1. Including and Linking SpRcApi

Four **SpRcApi** configurations are available for static linking: **SpRcApi.lib**, **SpRcApi(d).lib**, **SpRcApiMD.lib** and **SpRcApiMD(d).lib**.

The files with a lowercase 'd' at the end are the debug versions of the DTAPI library.

**SpRcApi(d).lib** has been compiled with the C/C++ Code-Generation Options in VC++ set to: "Use run-time library: Multithreaded." On the compiler command line this corresponds to the **/MT** option.

**SpRcApiMD(d).lib** has been compiled with the C/C++ Code-Generation Options in VC++ set to: "Use run-time library: Multithreaded DLL." On the compiler command line this corresponds to the **/MD** option.

The correct version of the **SpRcApi** library file will automatically be linked, because of a **pragma** directive in **SpRcApi.h**.

Automatic linking can be disabled by defining **\_SPRCAPI\_DISABLE\_AUTO\_LINK** (using **#define** or in the Pre-processor Definitions).

So, to use static link library of the **SpRcApi** follow these steps:

1. Copy **SpRcApi.h**, **DTAPI.h** and either **SpRcApi(d).lib** or **SpRcApiMD(d).lib** to your project or to a standard location visible to VC++.
2. Add **#include "SpRcApi.h"** to each file using **SpRcApi** functions.
3. Compile your application using the Multithreaded DLL (compiler switch **/MD**) or static (compiler switch **/MT**) version of the C run-time library.

#### Notes

- The static library files are available for VC10, VC11, VC12, VC14, VC15 and VC16.
- Using the release version of the static link library with a debug build of your main application may crash your application. This is caused by STL which uses different

lengths of data structures for debug and release builds.

### 3.2. Checking return codes

While using **SpRcApi**, it's important to check the return value after each call of an **SpRcApi** function. The connection to the StreamXpress playout server may get interrupted at any time, so each method call may fail. The exception to the rule is **GetVersion** which cannot fail.

For code clarity, the examples below do not check the return values of method calls. In production-quality code, however, it's essential to add such checks.

### 3.3. Connecting to StreamXpress

The following code establishes a connection to the StreamXpress.

```
// Create remote-control client
SpRcClient* SpRc;
SpRc = SpRcClient::CreateSpRcClient();

// Open a session
unsigned char Ip[] = {127,0,0,1};
SpRc->OpenSession(Ip, 9000);
```

Figure 1. Connecting to the StreamXpress.

The first step is to create an **SpRcClient** object, which represents the connection to the playout server.

The next step is to open a session with the playout server, using **OpenSession**. In this case, using local loopback address 127.0.0.1, a connection is established with the **StreamXpress** running on the same PC as the client application. Other PCs on the network can be reached by specifying their IP address.

### 3.4. Playing a file

The other methods in **SpRcApi** can be used to set parameters and play out a file. To a large extent they speak for themselves.

The code below disables looping, opens a file, starts playout and waits until playout is completed.

```
SpRc->SetLoopFlags(0);  
SpRc->OpenFile(L"C:\\Stream.ts");  
SpRc->SetPlayoutState(SPRC_STATE_PLAY);  
SpRc->WaitForCondition(  
    SPRC_COND_STOPPED, -1);
```

The file is opened in the context of the playout server, which means that C: is the C-disk on the playout server, not on the client.

The following code block plays a file for 10 seconds.

```
SpRc->SetPlayoutState(SPRC_STATE_PLAY);  
Sleep(10000);  
SpRc->SetPlayoutState(SPRC_STATE_STOP);
```

**SpRcClient – Session Interface****SpRcClient::CloseSession**

Close the session with the playout server.

```
SPRC_RESULT CloseSession();
```

**Parameters****Result**

SPRC_RESULT	Meaning
SPRC_E_COMMUNICATION	An error has occurred in the communication with the playout server
SPRC_OK	The session with the playout server has been closed successfully

**Remarks**



## SpRcClient::CreateSpRcClient

Create client object for issuing remote-control commands to a playout server.

```
static SpRcClient* CreateSpRcClient();
```

### Parameters

### Result

SPRC_RESULT	Meaning
NULL	Remote-control object cannot be created
pointer	Pointer to StreamXpress remote-control client

### Remarks

The SpRcApi is initialised with code similar to the following:

```
SpRcClient* SpRcApi = SpRcClient::CreateSpRcClient();
SPRC_RESULT Result = SpRcApi->OpenSession(IpAddr, PortNr);
```

After creating the client object and opening the session, all remote control commands are issued through the remote-control object **SpRcApi**.

## SpRcClient::GetRemoteVersion

Get **SpRcApi** version number as used in the playout server.

```
virtual void SpRcClient::GetRemoteVersion(
[out] int& Major           // Major version number
[out] int& Minor          // Minor version number
[out] int& BugFix         // Bug fix number
[out] int& Build          // Build number
);
```

### Parameters

*Major, Minor, BugFix, Build*

Version number of the **SpRcApi** library which was used in the StreamXpress. For an explanation of **SpRcApi** version numbering, see **SpRcClient::GetVersion**.

### Result

SPRC_RESULT	Meaning
SPRC_E_COMMUNICATION	An error has occurred in the communication with the playout server
SPRC_OK	The remote version number has been read successfully

### Remarks

## SpRcClient::GetRemoteDtapiVersion

Get DTAPI version number used to build StreamXpress.

```
virtual void SpRcClient::GetRemoteDtapiVersion(  
    [out] int& Major           // Major version number  
    [out] int& Minor          // Minor version number  
    [out] int& BugFix         // Bug fix number  
    [out] int& Build          // Build number  
);
```

### Parameters

*Major, Minor, BugFix, Build*

Version number of the DTAPI library which was used in the StreamXpress build.

### Result

SPRC_RESULT	Meaning
SPRC_E_COMMUNICATION	An error has occurred in the communication with the playout server
SPRC_OK	The remote Dtapi version number has been read successfully

### Remarks

## SpRcClient::GetVersion

Get version number of the **SpRcApi** client library.

```
virtual void SpRcClient::GetVersion(  
    [out] int& Major           // Major version number  
    [out] int& Minor          // Minor version number  
    [out] int& BugFix         // Bug fix number  
    [out] int& Build          // Build number  
);
```

### Parameters

#### *Major*

Major version number. This number is incremented when a non-backward compatible change is introduced in the StreamXpress remote-control API.

#### *Minor*

Minor version number. This number is incremented when a method is added to the StreamXpress remote-control API in a backward compatible way. For example, a client with version number 1.3.x.x will be able to interoperate with StreamXpress with API version 1.4.x.x.

#### *BugFix*

This number is incremented when a bug in the **SpRcApi** library has been fixed, without functional enhancements.

#### *Build*

The build number is a redundant version number that is incremented with every new version of the **SpRcApi** library.

### Result

SPRC_RESULT	Meaning
	No return value

### Remarks

## SpRcClient::OpenSession

Establish a session with the playout server.

```
SPRC_RESULT SpRcClient::OpenSession(  
    [in] unsigned char  IpAddr[4],    // IP address  
    [in] unsigned short PortNr       // Port number  
);
```

### Parameters

*IpAddr*

IP address of the playout server. If the StreamXpress is running on the same machine, 127.0.0.1

*PortNr*

Port number to access the playout server. The port number should match the port specified in the `-rc port` option when starting the StreamXpress.

### Result

SPRC_RESULT	Meaning
SPRC_E_NO_LICK	The port is not properly licensed for playout and remote control
SPRC_OK	The session with the playout server has been opened successfully
SPRC_VERSION_CONFLICT	A session with the playout server has been opened, but a version conflict has been detected between the version of the client <b>SpRcApi</b> and that of the server <b>SpRcApi</b>

### Remarks

The `SpRcClient` object supports a single session. If multiple sessions are required, multiple `SpRcClient` objects must be created.

## SpRcClient – Application Common Interface

### SpRcClient::GetAppInfo

Get information about the application.

```
virtual SPRC_RESULT SpRcClient::GetAppInfo(
    [out] std::wstring&  AppName,      // Application name
    [out] Int&  MajorVersion,        // Major version number
    [out] Int&  MinorVersion,       // Minor version number
    [out] Int&  BugFixVersion,      // Bug-fix version number
    [out] Int&  BuildNumber);      // Build number
);
```

#### Parameters

##### *AppName*

Application name as Unicode string. For the moment, the only application supporting SpRcApi is StreamXpress.

##### *MajorVersion*

Major version number. This number is incremented when the application implements major new functions.

##### *MinorVersion*

Minor version number. This number is incremented when the application implements small updates in functionality, possibly together with bug fixes.

##### *BugFixVersion*

Bug-fix version number. This number is incremented when the only changes relative to the last version of the application are bug fixes.

##### *BuildNumber*

Build number. This number is incremented with new builds of the application. It's never reset to zero so that each version of the application has a different build number.

#### Result

SPRC_RESULT	Meaning
SPRC_E_COMMUNICATION	An error has occurred in the communication with the playout server
SPRC_OK	Application information is returned successfully

#### Remarks

## SpRcClient::ShowWindow

Show or hide the *StreamXpress* application window.

```
virtual SPRC_RESULT SpRcClient::ShowWindow(  
    [in] bool Show           // Show or Hide  
);
```

### Parameters

*Show*

Show or hide the *StreamXpress* application window.

### Result

SPRC_RESULT	Meaning
SPRC_E_COMMUNICATION	An error has occurred in the communication with the playout server
SPRC_OK	The show or hide operation has been executed successfully

### Remarks

## SpRcClient – Port Selection Interface

### Struct PortDesc

Structure describing a physical playout port.

```
struct PortDesc {
    __int64 m_Serial;           // Unique serial number of the device
    int m_TypeNumber;         // Device type number
    int m_Ip[4];              // IP address (for IP ports only)
    int m_Mac [6];           // MAC address (for IP ports only)
    int m_FirmwareVersion;    // Firmware version
    int m_FirmwareVariant;    // Firmware variant
    int m_Port;               // Physical port number
    int m_OutputType;         // Output type (OR-able flags)
    int m_Capabilities;       // Capability flags (OR-able flags)
    int m_InUse;              // Output port already in use?
};

typedef std::vector<SpRcPortDesc> SpRcPortDescs;
typedef SpRcPortDescs::iterator SpRcPortDescIt;
```

### Members

*m\_Serial*

The serial number that uniquely identifies the DekTec device that hosts the playout port.

*m\_TypeNumber*

This integer corresponds to the number in the device's type number, e.g. 245 for the DTU-245.

*m\_Ip*

If the playout port is an IP-network port, this member identifies the IP address. Otherwise, the value of this member is undefined.

*m\_Mac*

If the playout port is an IP-network port, this member identifies the MAC address. Otherwise, the value of this member is undefined.

*m\_FirmwareVersion*

Version number of the firmware loaded on the device that hosts the playout port.

*m\_FirmwareVariant*

Variant of the firmware loaded on the device that hosts the playout port. Some DekTec devices may support multiple variants of the firmware each with different functionality.

*m\_Port*

This integer identifies the physical port number associated with this function. Please refer to DekTec's DTAPI documentation for an overview of physical port numbers per device.



### *m\_OutputType*

This field describes the type of stream that can be generated on this playout port. Output types are encoded in flags that may be OR-ed together to indicate that the port supports multiple types.

Value	Meaning
SPRC_OTYPE_ASI	DVB-ASI
SPRC_OTYPE_ATSC	ATSC (VSB) modulation
SPRC_OTYPE_CMMA	CMMA modulation
SPRC_OTYPE_DTMB	DTMB modulation
SPRC_OTYPE_DVBS	DVB-S modulation
SPRC_OTYPE_DVBS2	DVB-S.2 modulation
SPRC_OTYPE_DVBT	DVB-T modulation, includes DVB-H
SPRC_OTYPE_DVBT2	DVB-T2 modulation
SPRC_OTYPE_DVBT2MI	DVB-T2MI
SPRC_OTYPE_IQ	IQ samples
SPRC_OTYPE_ISDBS	ISDB-S modulation
SPRC_OTYPE_ISDBT	ISDB-T modulation
SPRC_OTYPE_QAM_A	QAM modulation, ITU-T J.83 Annex A (DVB-C)
SPRC_OTYPE_QAM_B	QAM modulation, ITU-T J.83 Annex B (US)
SPRC_OTYPE_QAM_C	QAM modulation, ITU-T J.83 Annex C (Japan)
SPRC_OTYPE_SDSDI	Standard-definition SDI
SPRC_OTYPE_SPI	DVB-SPI
SPRC_OTYPE_TSOIP	TS-over-IP
SPRC_OTYPE_ISDBS3	ISDB-S3 modulation
SPRC_OTYPE_DRM	DRM modulation
SPRC_OTYPE_ATSC3_STLTP	ATSC 3.0 STLTP modulation

### *m\_Capabilities*

This field describes further capabilities of the playout port. Capabilities are encoded in flags that may be OR-ed together to indicate that the port supports multiple capabilities.

Value	Meaning
<b>SPRC_ADJLVL</b>	Modulator port has an adjustable output level
<b>SPRC_CM</b>	Modulator port supports channel modelling
<b>SPRC_DIGIQ</b>	Modulator port has a digital IQ output
<b>SPRC_IF</b>	Modulator port has an IF output
<b>SPRC_LBAND</b>	Modulator port can upconvert to L-Band 950 .. 2150MHz
<b>SPRC_UHF</b>	Modulator port can upconvert to UHF Band 400 .. 862MHz
<b>SPRC_VHF</b>	Modulator port can upconvert to VHF Band 47 .. 470MHz

### *m\_InUse*

This status flag indicates whether the playout port is currently being used.

The "in-use" status is a snapshot of the current situation. Attaching to a playout port that is unused may fail because of race conditions with other applications.

Value	Meaning
<b>SPRC_PORT_CURR</b>	Port is the currently selected playout port in this remote-control session
<b>SPRC_PORT_UNUSED</b>	Port is not used
<b>SPRC_PORT_USED</b>	Port is used by another application

## SpRcClient::ScanPorts

Get information about the ports available for play out.

```
virtual SPRC_RESULT SpRcClient::ScanPorts(  
    [out] SpRcPortDesc& PortDescs // List of playout ports  
);
```

### Parameters

*PortDescs*

List of playout ports. Refer to **Struct PortDesc** for a description of attributes per port.

### Result

SPRC_RESULT	Meaning
SPRC_E_COMMUNICATION	An error has occurred in the communication with the playout server
SPRC_OK	Application information is returned successfully

### Remarks

## SpRcClient::SelectPort

Select a physical port for play out.

```
virtual SPRC_RESULT SpRcClient::SelectPort(
    [in] __int64 Serial,        // Serial number of device to be selected
    [in] int Port;            // Physical port number to be selected
    [in] int Modulation;      // Initial modulation standard
);
```

### Parameters

*Serial*

The serial number that identifies the DekTec device to be selected.

*Port*

Physical port number of port to be selected.

*Modulation*

For modulators only: Initial modulation standard. Use one of the `SPRC_MOD_XXX` constants. Set to 0 otherwise.

### Result

SPRC_RESULT	Meaning
SPRC_E_COMMUNICATION	An error has occurred in the communication with the playout server
SPRC_E_MOD_STANDARD	Initial modulation standard <i>Modulation</i> is not supported on the modulator port
SPRC_E_NO_LICK	The port is not properly licensed for playout and remote control
SPRC_E_NOT_FOUND	Cannot find the playout port identified by <i>Serial</i> and <i>Port</i>
SPRC_E_PORT_USED	The port could not be selected because it's in use by another instance of the StreamXpress or another application
SPRC_OK	The playout port has been selected successfully

### Remarks

The playout server will start without file selected and with parameters set to the defaults for the port.

A list of physical ports available for playout can be obtained with `SpRcClient::ScanPorts`.

## SpRcClient::SelectDtaPlus

Select a DtaPlus device to use as attenuator.

```
virtual SPRC_RESULT SpRcClient::SelectDtaPlus(  
    [in] bool UseDtaPlus,      // Whether a dta-plus should be used or not  
    [in] __int64 Serial,      // Serial number of DtaPlus to be selected  
);
```

### Parameters

*UseDtaPlus*

Set to true to actually start using a Dta-plus.

*Serial*

The serial number that identifies the DekTec device to be selected.

### Result

SPRC_RESULT	Meaning
SPRC_E_COMMUNICATION	An error has occurred in the communication with the playout server
SPRC_E_NOT_FOUND	Cannot find the Dta-plus port identified by <i>Serial</i>
SPRC_E_PORT_USED	The Dta-plus could not be selected because it's in use by another instance of the StreamXpress or another application
SPRC_OK	The Dta-plus port has been selected successfully

### Remarks

## SpRcClient – Payout Interface

### Struct SpRcAsiPars

Payout parameters for a DVB-ASI output port.

```

struct SpRcAsiPars {
    bool   m_Remux;           // Remultiplex yes/no
    int    m_PlayoutRate;    // Only used if Remux is on
    bool   m_BurstMode;     // DVB-ASI burst mode
    int    m_TxMode;        // Transmit mode
    int    m_Polarity;     // Physical polarity of the ASI signal
};

```

### Members

*m\_Remux*

Turn remultiplexing on (true) or off (false). If remultiplexing is on, the StreamXpress adds null packets and adjusts timing information so that the stream is played out at *m\_PlayoutRate*. If remultiplexing is off, the stream is played out at the Transport-Stream rate, and *m\_Playout* is not used.

*m\_PlayoutRate*

DVB-ASI payout rate.

*m\_BurstMode*

Turn DVB-ASI burst mode on (true) or off (false).

*m\_TxMode*

Transmit mode.

Value	Meaning
DTAPI_TXMODE_188	Transport Packets are assumed to be 188 bytes, and are played out as 188-bytes packets.
DTAPI_TXMODE_204	Transport Packets are assumed to be 204 bytes, and are played out as 204-bytes packets.
DTAPI_TXMODE_ADD16	Transport Packets are assumed to be 188 bytes, and are played out as 204-bytes packets.
DTAPI_TXMODE_MIN16	Transport Packets are assumed to be 204 bytes, and are played out as 188-bytes packets.
DTAPI_TXMODE_RAW	No assumptions are made on packet structure. Bytes in the stream are transmitted unmodified. Null-packet stuffing cannot be applied.

*m\_Polarity*

Polarity of the DVB-ASI signal.

Value	Meaning
DTAPI_TXPOL_NORMAL	Generate a 'normal' ASI signal
DTAPI_TXPOL_INVERTED	Generate an inverted ASI signal

## Struct SpRcCmmbPars

CMMB parameters.

```
struct SpRcCmmbPars {  
    int m_Bandwidth;           // CMMB Bandwidth  
    int m_AreaId;             // Area ID (0..127)  
    int m_TxId;               // Transmitter ID (128..255)  
};
```

### Members

*m\_Bandwidth*

CMMB Bandwidth.

Value	Meaning
DTAPI_CMMB_BW_2MHZ	CMMB 2MHz Bandwidth
DTAPI_CMMB_BW_8MHZ	CMMB 8MHz Bandwidth

*m\_AreaId*

Area ID (0..127).

*m\_TxId*

Transmitter ID (128..255).



## Struct SpRcCmPars

Channel-modelling parameters.

```
struct SpRcCmPars {  
    bool m_CmEnable;           // Enable channel modelling  
    bool m_AwgnEnable;        // Enable noise injection  
    bool m_PathsEnable;       // Enable transmission path simulation  
    double m_Snr;             // Signal-to-noise ratio in dB  
    std::vector<SpRcCmPath> m_Paths; // List of transmission paths  
};
```

### Members

*m\_CmEnable*

If true, perform channel modelling according to the other values in this struct.  
If false, channel modelling is disabled and the other struct values are ignored.

*m\_AwgnEnable*

Enable white noise injection; *m\_Snr* specifies the signal to noise ratio.

*m\_PathsEnable*

Enable multipath echo simulation.

*m\_Snr*

The noise power is defined relative to an imaginative 0dB output signal of the modulator. This means that *m\_Snr* is the real signal-to-noise ratio only if the accumulated power of the paths in *m\_Paths* is 0dB.

*m\_Paths*

List of transmission-path descriptions (maximum 32). See struct **SpRcCmPath**.

## Struct SpRcCmPaths

Channel-modelling parameters for a single .

```

struct SpRcCmPath {
    int   m_Type;           // Type of path fading
    double m_Attenuation;  // Attenuation in dB
    double m_Delay;       // Delay in us
    double m_Phase;       // Phase shift in degrees
    double m_Doppler;     // Doppler Frequency in Hz
};

```

### Members

*m\_Type*

Type of transmission path values:

Value	Meaning
SPRC_CONSTANT_DELAY	Constant delay and phase
SPRC_CONSTANT_DOPPLER	Constant frequency shift
SPRC_RAYLEIGH_JAKES	Raleigh fading with Jakes power spectral density (mobile path model)
SPRC_RAYLEIGH_GAUSSIAN	Raleigh fading with Gaussian power spectral density (ionspheric path model)

*m\_Attenuation*

Attenuation in dB. The total attenuation of all paths may not exceed 0dB to avoid overflow of the channel simulator.

*m\_Delay*

Delay in us. The maximum delay for an 8MHz channel is 896us.

*m\_Phase*

Phase shift in degrees (value ignored for type **RAYLEIGH\_JAKES** and **RAYLEIGH\_GAUSSIAN**).

*m\_Doppler*

Doppler frequency in Hz (value ignored for type **CONSTANT\_DELAY**). The corresponding Speed in m/s is:  $Speed = f_{doppler} * 3.10^8 / f_{RF}$ .

## Struct SpRcDvbT2Group

Parameters to specify standard DVB-T2 parameter set groups

```
struct SpRcDvbT2Group {  
    std::wstring m_GroupName;    // Name of the DVB-T2 group  
    std::wstring m_GroupRefName; // Specific set in group  
};
```

### Members

*m\_GroupName*

Name of the DVB-T2 group, e.g. "VV1xx".

*m\_GroupRefName*

Specific set in group, e.g. "VV100".

## Struct SpRcDvbT2Pars

Structure describing DVB-T2 modulation parameters.

```

struct SpRcDvbT2Pars {
    int m_T2Version; // DVB-T2 version DTAPI_DVBT2_VERSION_x
    int m_Bandwidth; // Channel bandwidth DTAPI_DVBT2_8MHZ/...
    int m_FftMode; // FFT mode (or size) DTAPI_DVBT2_FFT_x
    int m_Miso; // MISO mode DTAPI_DVBT2_MISO_x
    int m_GuardInterval; // Guard interval DTAPI_DVBT2_GI_x
    int m_Papr; // PAPR reduction mode DTAPI_DVBT2_PAPR_x
    int m_BwtExt; // Bandwidth extension 0 or 1
    int m_PilotPattern; // Pilot pattern 1 to 8
    int m_NumT2Frames; // Number of T2 frames in a super frame
    int m_NumDataSyms; // Number of data OFDM symbols per T2 frame
    int m_L1Modulation; // L1 modulation type DTAPI_DVBT2_BPSK/...
    bool m_FefEnable; // Insert FEF (yes/no)
    int m_FefType; // FEF type 0 ... 15
    int m_FefLength; // FEF length
    int m_FefS1; // FEF S1 field value 2 <= FefS1 <= 7
    int m_FefS2; // FEF S2 field value 0 <= FefS2 <= 15
    int m_FefInterval; // FEF interval
    int m_FefSignal; // Type of signal during FEF period
    int m_CellId; // Cell ID
    int m_NetworkId; // Network ID
    int m_T2SystemId; // T2 system ID
    int m_Frequency; // L1-post frequency field value
    // PLP#0 parameters
    bool m_Hem; // High Efficiency Mode (yes/no)
    bool m_Npd; // Null Packet Deletion (yes/no)
    bool m_IssyEnabled; // ISSY enabled (yes/no)
    int m_Id; // PLP ID
    int m_GroupId; // PLP group ID
    int m_Type; // PLP type DTAPI_DVBT2_PLP_TYPE_x
    int m_CodeRate; // Code rate
    int m_Modulation; // Modulation type DTAPI_DVBT2_BPSK/...
    bool m_Rotation; // Constellation rotation (yes/no)
    int m_FecType; // FEC type 0=LDPC 16K, 1=LDPC 64K
    int m_TimeIlLength; // Time interleaving length 0..255
    int m_TimeIlType; // Timer interleaving type 0 or 1
    bool m_InBandFlag; // In band signalling information (yes/no)
    bool m_NumBlocks; // Number of FEC blocks per IL frame
    int m_FollowMode; // Mode to compute NumDataSyms/NumBlocks
};

```

### Members

*m\_T2Version*, *m\_Bandwidth*, *m\_FftMode*, *m\_Miso*, *m\_GuardInterval*, *m\_Papr*, *m\_BwtExt*, *m\_PilotPattern*, *m\_NumT2Frames*, *m\_NumDataSyms*, *m\_L1Modulation*, *m\_FefEnable*, *m\_FefType*, *m\_FefLength*, *m\_FefS1*, *m\_FefS2*, *m\_FefInterval*, *m\_FefSignal*, *m\_CellId*, *m\_NetworkId*, *m\_T2SystemId*, *m\_Frequency*

Overall DVB-T2 modulation parameters that are not specific per PLP. For a description of these parameters, please refer to the section describing **DtDvbT2Pars** in the DTAPI specification.

*m\_Hem, m\_Npd, m\_IssyEnabled, m\_Id, m\_GroupId, m\_Type, m\_CodeRate, m\_Modulation, m\_Rotation, m\_FecType, m\_TimeI1Length, m\_TimeI1Type, m\_InBandFlag, m\_NumBlocks*

Parameters for PLP 0 (the one and only PLP). For a description of these parameters, please refer to the section describing **struct DtDvbT2PlpPars** in the DTAPI specification.

*m\_FollowMode*

This parameter specifies how StreamXpress computes or copies **NUM\_DATA\_SYMBOLS** (the number of data OFDM symbols per T2 frame) and **PLP\_NUM\_BLOCKS** (the number of FEC blocks per interleaving frame.)

Value	Meaning
<b>SPRC_T2_FOLLOW_OFF</b>	No automatic computations. <b>NUM_DATA_SYMBOLS</b> is copied from <i>m_NumDataSyms</i> , while <b>PLP_NUM_BLOCKS</b> is copied from <i>m_NumBlocks</i>
<b>SPRC_T2_FOLLOW_OPT1</b>	Compute <b>NUM_DATA_SYMBOLS</b> and <b>PLP_NUM_BLOCKS</b> . Given the DVB-T2 modulation parameters in <b>SpRcDvbT2Pars</b> , the StreamXpress optimises both <b>NUM_DATA_SYMBOLS</b> and <b>PLP_NUM_BLOCKS</b> . The values in <i>m_NumDataSyms</i> and <i>m_NumBlocks</i> are not used.
<b>SPRC_T2_FOLLOW_OPT2</b>	Compute <b>PLP_NUM_BLOCKS</b> . Given the DVB-T2 modulation parameters in <b>SpRcDvbT2Pars</b> including <i>m_NumDataSyms</i> , the StreamXpress optimises <b>PLP_NUM_BLOCKS</b> . The value in <i>m_NumBlocks</i> is not used. <b>NUM_DATA_SYMBOLS</b> is copied from <i>m_NumDataSyms</i> .

## Struct SpRcHwNoisePars

Noise parameters for modulators DTA-107 and DTA-2107.

```
struct SpRcHwNoisePars {  
    bool    m_SnrOn;           // Enable Noise generator  
    double  m_Snr;           // Signal Noise ratio  
};
```

### Members

*m\_SnrOn*

Enable Noise generator.

*m\_Snr*

Signal-to-noise ratio in dB.

## Struct SpRcIsdbtLayerPars

Structure describing ISDB-T modulation parameters for one hierarchical layer. This structure is used in `SpRcIsdbtPars`, in an array of three structs for layer A, B and C.

```

struct SpRcIsdbtLayerPars {
    int m_NumSegments; // Number of segments
    int m_Modulation; // Modulation type
    int m_CodeRate; // Code rate
    int m_TimeInterleave; // Time interleaving
};
    
```

### Members

`m_NumSegments`

Number of segments used in this layer. The sum of `m_NumSegment` must be 13.

`m_Modulation`

Modulation type applied to the segments in this layer.

Value	Meaning
DTAPI_ISDBT_MOD_DQPSK	DQPSK
DTAPI_ISDBT_MOD_QPSK	QPSK
DTAPI_ISDBT_MOD_QAM16	16-QAM
DTAPI_ISDBT_MOD_QAM64	64-QAM

`m_CodeRate`

Convolutional coding rate applied to the segments in this layer.

Value	Meaning
DTAPI_ISDBT_RATE_1_2	1/2
DTAPI_ISDBT_RATE_2_3	2/3
DTAPI_ISDBT_RATE_3_4	3/4
DTAPI_ISDBT_RATE_5_6	5/6
DTAPI_ISDBT_RATE_7_8	7/8

`m_TimeInterleave`

Encoded length of time interleaving.

The table below defines the mapping of  $m\_TimeInterleave$  to parameter  $l$  in the time-interleaving process.

Value	Mode 1	Mode 2	Mode 3
0	0	0	0
1	4	2	1
2	8	4	2
3	16	8	4



## Struct SpRcIsdbtPars

Modulation parameters for ISDB-T.

```

struct SpRcIsdbtPars {
    bool m_DoMux; // Hierarchical multiplexing yes/no
    int m_BType; // Broadcast type
    int m_Mode; // Transmission mode
    int m_Guard; // Guard interval
    int m_PartialRx; // Partial reception
    int m_Emergency; // Switch-on control for emergency broadcast
    int m_IipPid; // PID used for multiplexing IIP packet
    SpRcIsdbtLayerPars m_LayerPars[3]; // Layer-A/B/C parameters
    std::map<int, int> m_Pid2Layer; // PID-to-layer map
    int m_LayerOther; // Other PIDs are mapped to this layer
    int m_ParXtra0; // Extra parameters
    int m_Virtual13Segm; // Virtual 13-segment mode
};

struct SpRcIsdbtLayerPars {
    int m_NumSegment; // Number of segments
    int m_Modulation; // Modulation type
    int m_CodeRate; // Code rate
    int m_TimeInterleave; // Time interleaving
};

```

### Members

*m\_DoMux*

If true, perform hierarchical multiplexing in accordance with the ISDB-T parameters as defined explicitly in this structure.

If false, the ISDB-T modulation parameters are specified indirectly by the TMCC information in the 16 extra bytes of the 204-byte packets.

*m\_BType*

Broadcast type.

Value	Meaning
DTAPI_ISDBT_BTYPE_TV	TV broadcast; Can be used with any number of segments
DTAPI_ISDBT_BTYPE_RAD1	1-segment radio broadcast; Total #segments must be 1
DTAPI_ISDBT_BTYPE_RAD3	3-segment radio broadcast; Total #segments must be 3

*m\_Mode*

Transmission mode.

Value	Meaning
1	Mode 1: 2k
2	Mode 2: 4k
3	Mode 3: 8k

*m\_Guard*

Guard-interval length.

Value	Meaning
DTAPI_ISDBT_GUARD_1_32	1/32
DTAPI_ISDBT_GUARD_1_16	1/16
DTAPI_ISDBT_GUARD_1_8	1/8
DTAPI_ISDBT_GUARD_1_4	1/4

*m\_PartialRx*

Flag that indicates whether layer A is used for partial reception: 0 = no partial reception, 1 = partial reception on.

*m\_Emergency*

Flag that indicates whether the switch-on control flag for emergency broadcast should be turned on: 0 = off, 1 = on.

*m\_IipPid*

PID value used for multiplexing the IIP packet.

*m\_LayerPars*

Modulation parameters for hierarchical layers A (element 0), B (1) and C (2).

*m\_Pid2Layer*

Map that specifies the hierarchical layer, or layers, to which an elementary stream is to be mapped. The key in the map is the PID of the elementary stream. The value stored in the map is an OR of one or more flags listed in the table below. A value of 0 indicates that the elementary stream is to be dropped.

Value	Meaning
DTAPI_ISDBT_LAYER_A	Map elementary stream to layer A
DTAPI_ISDBT_LAYER_B	Map elementary stream to layer B
DTAPI_ISDBT_LAYER_C	Map elementary stream to layer C

*m\_LayerOther*

Map streams with PIDs not in *m\_Pid2Layer* to this layer.

*m\_ParXtra0*

Extra parameter encoding bandwidth, sample rate and number of segments. This parameter is encoded like ParXtra0 in SetModControl with *ModType* DTAPI\_MOD\_ISDBT.

*m\_Virtual13Segm*

Use virtual 13 segment mode. The number of segments in layer B is "faked" to be 12.

## Struct SpRcModPars

Modulation parameters for all modulation standards except DVB-T2 and ISDB-T.

```
struct SpRcModPars {  
    int m_ModType;           // Modulation type  
    int m_ParXtra0;         // Extra modulation parameter 0  
    int m_ParXtra1;         // Extra modulation parameter 1  
    int m_ParXtra2;         // Extra modulation parameter 2  
    int m_SymRate;          // Symbol rate in bd  
};
```

### Members

*m\_ModType*

Modulation type, see `DTAPI_MOD_XXX` constants.

*m\_ParXtra0*, *m\_ParXtra1*, *m\_ParXtra2*

Modulation parameters, see DTAPI documentation (`SetModControl`).

*m\_SymRate*

Symbol rate in baud. Required for modulation standards that require a symbol rate, like DVB-S. This member should be set to -1 if no symbol rate is required.

## Struct SpRcPlayoutInfo

Structure describing static playout information.

```

struct SpRcPlayoutInfo {
    bool m_BurstMode;           // DVB-ASI burst mode
    bool m_ExtClock;           // Use external clock
    bool m_FileCanBeRead;      // A file has been selected that can be read
    std::wstring m_Filename;   // Currently selected filename
    long m_FileOffsetEnd;      // Number of unused bytes at end of file
    long m_FileOffsetStart;    // Number of unused bytes at start of file
    long m_FilePlayedBytes;    // File length minus bytes at start and end
    int m_FileRateEst;         // TS: Estimated file rate
    long m_FileSize;           // Size of the file
    int m_FileType;            // Type of data in file: RAW/TS/SDI
    double m_LoopBeginRel;     // Subloop, begin position (relative 0..1)
    double m_LoopEndRel;       // Subloop, end position (relative 0..1)
    int m_LoopFlags;           // Adapt CC/PCR/TDT and wrap-around flags
    int m_PlayoutState;        // HOLD/PLAYING
    int m_PlayoutRate;         // Playout rate @188
    boolean m_Remux;           // Remultiplex mode
    int m_SymRate;             // Modulators: Symbol rate
    double m_TimeLoopBegin;    // Time corresponding to beginning of loop
    double m_TimeLoopEnd;      // Time corresponding to end of loop
    int m_TimeOffset;          // Offset added to playout time
    int m_TsRate;              // TS: TS rate @188
    int m_TpSize;              // TS: packet size
    int m_TxPolarity;          // Transmit polarity for ASI channels
};

```

### Members

*m\_BurstMode*

For ASI channels: DVB-ASI is sent in burst mode yes/no.

*m\_ExtClock*

The transport-stream is taken from the external-clock input.

*m\_FileCanBeRead*

The file that is currently selected is valid.

*m\_Filename*

The name of the file that is currently selected in *StreamXpress*.

*m\_FileOffsetEnd*

Number of unused bytes at the end of the file.

*m\_FileOffsetStart*

Number of unused bytes at start of the file.

*m\_FilePlayedBytes*

The number of bytes from the file that are actually played out (file length minus number of unused bytes at start and end).

*m\_FileRateEst*

TS: Estimated file rate.

*m\_FileSize*

Size of the file.

*m\_FileType*

Type of data in the file: raw data, transport stream or SDI.

*m\_LoopBeginRel*

Start position of the subloop as a relative number between 0 and 1. If a subloop is not used, the value is 0.

*m\_LoopEndRel*

End position of the subloop as a relative number between 0 and 1. If a subloop is not used, the value is 0.

*m\_LoopFlags*

Loop adaptation flags: Adaptation of CC, PCR, TDT and wrap-around flags.

*m\_PlayoutState*

Indicates whether the StreamXpress is paused (hold mode) or playing.

*m\_PlayoutRate*

Rate at which the stream is played. For transport streams, the rate at 188 bytes per packet is used, even if the stream contains 204-byte transport packets.

*m\_Remux*

For transport streams only: Indicates whether the transport-stream is re-multiplexed to another rate.

*m\_SymRate*

For modulators only: Symbol rate.

*m\_TimeLoopBegin*

Playout time corresponding to the beginning of the loop.

*m\_TimeLoopEnd*

Playout time corresponding to the end of loop.

*m\_TimeOffset*

Offset that is added to the playout time.

*m\_TsRate*

Rate of the transport stream in the. For transport streams, the rate at 188 bytes per packet is used, even if the file contains 204-byte transport packets.

*m\_TpSize*

For transport streams only: size of the transport packets.

*m\_TxPolarity*

Transmit polarity for ASI channels.

## Struct SpRcPlayoutStatus

Structure describing the dynamic playout status.

```
struct SpRcPlayoutInfo {  
    int   m_FifoLoad;           // Current FIFO load  
    int   m_NumErrors;         // Number of errors (underflows)  
    int   m_NumWraps;          // #wraps  
    double m_PosRel;           // Relative position in subloop (0..1)  
    int   m_TotalMemLoad;      // #bytes in DiskBuffer+MemBuffer (snapshot)  
};
```

### Members

*m\_FifoLoad*

Current load of the output FIFO.

*m\_NumErrors*

Number of underflow errors since the last start of playout.

*m\_NumWraps*

Number of times that the file has wrapped since the last start of playout.

*m\_PosRel*

Relative position in the subloop. The client can use *m\_PosRel* to synchronise a slider to the slider in the **StreamXpress**.

*m\_TotalMemLoad*

Snapshot of the number of data bytes stored in the disk and memory buffer.

## Struct SpRcRfPars

Structure describing RF parameters for modulators.

```
struct SpRcRfPars {  
    __int64 m_Frequency;    // RF frequency (Hz)  
    double m_Level;        // RF output level (dBm)  
    bool m_SpecInv;        // RF Spectral inversion  
    bool m_CW;             // RF CW mode  
    bool m_RfEnabledOnStop; // RF output enabled on stop  
};
```

### Members

*m\_Frequency*

Center frequency in Hz of the upconverted signal.

*m\_Level*

Level of the main output signal in dBm.

*m\_SpecInv*

Spectral inversion.

*m\_CW*

CW.

*m\_RfEnabledOnStop*

Whether or not the RF output is muted when playback is stopped.

## Struct SpRcSpiPars

Playout parameters for a DVB-SPI output port.

```

struct SpRcSpiPars {
    bool   m_Remux;           // Remultiplex yes/no
    int   m_PlayoutRate;    // Only used if Remux is on
    int   m_TxMode;         // Transmit mode
    bool   m_Power;         // Turn power on/off for external adapter
};

```

### Members

*m\_Remux*

Turn remultiplexing on (true) or off (false). If remultiplexing is on, the StreamXpress adds null packets and adjusts timing information so that the stream is played out at *m\_PlayoutRate*. If remultiplexing is off, the stream is played out at the Transport-Stream rate, and *m\_Playout* is not used.

*m\_PlayoutRate*

DVB-SPI playout rate.

*m\_TxMode*

Transmit mode.

Value	Meaning
DTAPI_TXMODE_188	Transport Packets are assumed to be 188 bytes, and are played out as 188-bytes packets.
DTAPI_TXMODE_192	192-byte mode (DTA-102 only) Transport Packets are assumed to be 192 bytes, and are played out as 192-bytes packets.
DTAPI_TXMODE_204	Transport Packets are assumed to be 204 bytes, and are played out as 204-bytes packets.
DTAPI_TXMODE_ADD16	Transport Packets are assumed to be 188 bytes, and are played out as 204-bytes packets.
DTAPI_TXMODE_MIN16	Transport Packets are assumed to be 204 bytes, and are played out as 188-bytes packets.
DTAPI_TXMODE_RAW	No assumptions are made on packet structure. Bytes in the stream are transmitted unmodified. Null-packet stuffing cannot be applied.

*m\_Power*

Power for external adapter is switched on or off.



## Struct SpRcTsgPars

Parameters for the test-signal generator.

```
struct SpRcTsgPars {  
    int m_Type;           // Type of signal generator to use  
    int m_Pid;           // PID to carry the signal for TS modes  
    int m_VidStd;        // Video standard for SDI modes  
    int m_Flags;         // Reserved, set to 0  
};
```

### Members

*m\_Type*

Type of signal generator to use:

Value	Meaning
SPRC_TSG_TYPE_PRBS7	PRBS-7 TS generator
SPRC_TSG_TYPE_PRBS15	PRBS-15 TS generator
SPRC_TSG_TYPE_PRBS23	PRBS-23 TS generator
SPRC_TSG_TYPE_PRBS31	PRBS-31 TS generator
SPRC_TSG_TYPE_SDI	SDI generator

*m\_Pid*

PID to carry the PRBS packets. Ignored while in SDI mode.

*m\_VidStd*

Video standard used for SDI signal generator. Ignored while in transport-stream mode.

Value	Meaning
SPRC_VIDSTD_525I59_94	
SPRC_VIDSTD_625I50	
SPRC_VIDSTD_720P23_98	
SPRC_VIDSTD_720P24	
SPRC_VIDSTD_720P25	
SPRC_VIDSTD_720P29_97	
SPRC_VIDSTD_720P30	
SPRC_VIDSTD_720P50	
SPRC_VIDSTD_720P59_94	
SPRC_VIDSTD_720P60	
SPRC_VIDSTD_1080I50	
SPRC_VIDSTD_1080I59_94	
SPRC_VIDSTD_1080I60	
SPRC_VIDSTD_1080P23_98	
SPRC_VIDSTD_1080P24	
SPRC_VIDSTD_1080P25	
SPRC_VIDSTD_1080P29_97	
SPRC_VIDSTD_1080P30	
SPRC_VIDSTD_1080P50	
SPRC_VIDSTD_1080P59_94	
SPRC_VIDSTD_1080P60	

*m\_Flags*

Reserved, set to 0.

## Struct SpRcTsoipPars

Parameters for a Transport-Stream-over-IP port.

```

struct SpRcTsoipPars {
    int m_TxMode; // Transmission mode (188, 204, Add16, ...)
    unsigned char m_Ip[4]; // IP address
    int m_Port; // Port number
    bool m_EnaFallover; // Enable IP double-buffering
    unsigned char m_Ip2[4]; // 2nd IP address, used for double-buffering
    int m_Port2; // 2nd port number, used for double-buffering
    int m_TimeToLive; // TTL
    int m_NumTpPerIp; // #TPs per IP packet
    int m_Protocol; // Protocol: UDP/RTP
    int m_DiffServ; // Differentiated services
    int m_FecMode; // Error correction mode
    int m_FecNumRows; // 'D' = #rows in FEC matrix
    int m_FecNumCols; // 'L' = #columns in FEC matrix
};

```

### Members

*m\_TxMode*

Transmit mode.

Value	Meaning
DTAPI_TXMODE_188	Transport Packets are assumed to be 188 bytes, and are played out as 188-bytes packets.
DTAPI_TXMODE_204	Transport Packets are assumed to be 204 bytes, and are played out as 204-bytes packets.
DTAPI_TXMODE_ADD16	Transport Packets are assumed to be 188 bytes, and are played out as 204-bytes packets.
DTAPI_TXMODE_MIN16	Transport Packets are assumed to be 204 bytes, and are played out as 188-bytes packets.

*m\_Ip[4]*

Destination IP address. If the IP address is in the multicast range, the playout server automatically joins and drops membership of the multicast group.

*m\_Port*

Destination port number.

*m\_EnaFallover*

Enable IP double-buffering. Currently only supported on the DTA-2162. The second IP port will playout a duplicate of the signal which can be used by a receiver to conceal errors in either of the two transmission paths.

*m\_Ip2[4]*

*m\_Port2*

See *m\_Ip* and *m\_Port*. Used if *m\_EnaFallover* is true to configure the second IP port.

*m\_TimeToLive*

Time-To-Live (TTL) value to be used for multicast transmission. When *m\_Ttl* is 0, a default value is used.

*m\_NumTpPerIp*

Number of Transport Packets (TPs) stored in one IP packet. The range is 1..7.

*m\_Protocol*

Protocol expected for encapsulation of Transport Packets.

Value	Meaning
DTAPI_PROTO_UDP	UDP
DTAPI_PROTO_RTP	RTP

*m\_DiffServ*

Value to be put in the *Differentiated Services* field (formerly *Service Type*) in the IP header.

*m\_FecMode*

Error-correction mode.

Value	Meaning
DTAPI_FEC_DISABLE	No FEC
DTAPI_FEC_2D	RFC2733 parity FEC with 2D extensions as described in Code of Practice #3

*m\_FecNumRows*, *m\_FecNumCols*

Number of rows and columns in the FEC matrix. In the *COP #3* these parameters are called *D* and *L* respectively. The following restrictions apply to *L* and *D*:

$$4 \leq D \leq 20, \quad 1 \leq L \leq 20 \quad \text{and} \quad L * D \leq 100$$

## SpRcClient::ClearErrors

Clear number-of-errors counter.

```
virtual SPRC_RESULT SpRcClient::ClearErrors(  
);
```

### Parameters

### Result

SPRC_RESULT	Meaning
SPRC_E_COMMUNICATION	An error has occurred in the communication with the playout server
SPRC_E_NO_PORT	Invalid operation because no port is selected
SPRC_OK	The DVB-ASI transmission parameters have been read successfully

### Remarks

## SpRcClient::GetAsiPars

Get DVB-ASI transmission parameters.

```
virtual SPROC_RESULT SpRcClient::GetAsiPars (
    [out] SpRcAsiPars& AsiPars // ASI parameters
);
```

### Parameters

*AsiPars*

DVB-ASI transmission parameters, see **Struct SpRcAsiPars**.

### Result

SPRC_RESULT	Meaning
SPRC_E_COMMUNICATION	An error has occurred in the communication with the playout server
SPRC_E_NO_PORT	Invalid operation because no port is selected
SPRC_E_NOT_ASI	Invalid operation because the port is not an ASI port or is operating in SDI mode
SPRC_OK	The DVB-ASI transmission parameters have been read successfully

### Remarks

## SpRcClient::GetChannelModellingPars

Get Channel Modelling parameters.

```
virtual SPRC_RESULT SpRcClient::GetChannelModellingPars (  
    [out] SpRcCmPars& CmPars           // Channel Modelling parameters  
);
```

### Parameters

*CmPars*

Channel Modelling parameters, see **Struct SpRcCmPars**.

### Result

SPRC_RESULT	Meaning
SPRC_E_COMMUNICATION	An error has occurred in the communication with the playout server
SPRC_E_NO_PORT	Invalid operation because no port is selected
SPRC_E_NOT_MOD	Invalid operation because the port is not a modulation port
SPRC_OK	The Channel Modelling parameters have been read successfully

### Remarks

## SpRcClient::GetCmmbPars

Get CMMB parameters.

```
virtual SPRC_RESULT SpRcClient::GetCmmbPars (  
    [out] SpRcCmmbPars& CmmbPars // CMMB parameters  
);
```

### Parameters

*CmmbPars*

CMMB parameters, see **Struct SpRcCmmbPars**.

### Result

SPRC_RESULT	Meaning
SPRC_E_COMMUNICATION	An error has occurred in the communication with the playout server
SPRC_E_NO_PORT	Invalid operation because no port is selected
SPRC_E_NOT_CMMB	Invalid operation because the CMMB modulation is not configured
SPRC_E_NOT_MOD	Invalid operation because the port is not a modulation port
SPRC_OK	The parameters have been read successfully

### Remarks



## SpRcClient::GetDvbT2Group

Get current DVB-T2 group selection.

```
virtual SPRC_RESULT SpRcClient::GetDvbT2Group(  
    [out] SpRcDvbT2Group& DvbT2Group // DVB-T2 group  
);
```

### Parameters

*DvbT2Group*

DVB-T2 group, see Struct `SpRcDvbT2Group`.

### Result

SPRC_RESULT	Meaning
SPRC_E_COMMUNICATION	An error has occurred in the communication with the playout server
SPRC_E_NO_PORT	Invalid operation because no port is selected
SPRC_E_NOT_DVBT2	Invalid operation because the DVB-T2 modulation is not configured
SPRC_E_NOT_MOD	Invalid operation because the port is not a modulation port
SPRC_OK	The parameters have been read successfully

### Remarks

## SpRcClient::GetDvbT2Pars

Get DVB-T2 parameters.

```
virtual SPRC_RESULT SpRcClient::GetDvbT2Group(  
    [out] SpRcDvbT2Pars& DvbT2Pars // DVB-T2 parameters  
);
```

### Parameters

*DvbT2Pars*

DVB-T2 parameters, see **Struct SpRcDvbT2Pars**.

### Result

SPRC_RESULT	Meaning
SPRC_E_COMMUNICATION	An error has occurred in the communication with the playout server
SPRC_E_NO_PORT	Invalid operation because no port is selected
SPRC_E_NOT_DVBT2	Invalid operation because the DVB-T2 modulation is not configured
SPRC_E_NOT_MOD	Invalid operation because the port is not a modulation port
SPRC_OK	The parameters have been read successfully

### Remarks

## SpRcClient::GetHwNoisePars

Get Noise parameters for modulators DTA-107 and DTA-2107.

```
virtual SPRC_RESULT SpRcClient::GetHwNoisePars (  
    [out] SpRcHwNoisePars& HwNoisePars // Hw Noise parameters  
);
```

### Parameters

*HwNoisePars*

Noise parameters, see Struct `SpRcHwNoisePars`.

### Result

SPRC_RESULT	Meaning
SPRC_E_COMMUNICATION	An error has occurred in the communication with the playout server
SPRC_E_NO_PORT	Invalid operation because no port is selected
SPRC_E_NOT_MOD	Invalid operation because the port is not a modulation port
SPRC_OK	The parameters have been read successfully

### Remarks

## SpRcClient::GetIsdbtPars

Get ISDB-T modulation parameters.

```
virtual SPROC_RESULT SpRcClient::GetIsdbtPars(  
    [out] SpRcIsdbtPars& IsdbtPars // ISDB-T parameters  
);
```

### Parameters

*IsdbtPars*

ISDB-T modulation parameters, see **Struct SpRcIsdbtPars**.

### Result

SPRC_RESULT	Meaning
SPRC_E_COMMUNICATION	An error has occurred in the communication with the playout server
SPRC_E_NO_PORT	Invalid operation because no port is selected
SPRC_E_NOT_ISDBT	Invalid operation because the port is operating in ISDB-T modulation mode
SPRC_E_NOT_MOD	Invalid operation because the port is not a modulation port
SPRC_OK	The modulation parameters have been read successfully

### Remarks

## SpRcClient::GetIqGain

Get IQ gain parameter.

```
virtual SPROC_RESULT SpRcClient::GetIqGain(  
    [out] int& IqGain // IQ gain  
);
```

### Parameters

*IqGain*

The gain of the IQ signal.

### Result

SPRC_RESULT	Meaning
SPRC_E_COMMUNICATION	An error has occurred in the communication with the playout server
SPRC_E_NO_PORT	Invalid operation because no port is selected
SPRC_E_NOT_MOD	Invalid operation because the port is not a modulation port
SPRC_OK	The modulation parameters have been read successfully

### Remarks

## SpRcClient::GetModPars

Get modulation parameters.

```
virtual SPRC_RESULT SpRcClient::GetModPars (  
    [out] SpRcModPars& ModPars // Modulation parameters  
);
```

### Parameters

*ModPars*

Modulation parameters, see **Struct SpRcModPars**.

### Result

SPRC_RESULT	Meaning
SPRC_E_COMMUNICATION	An error has occurred in the communication with the playout server
SPRC_E_NO_PORT	Invalid operation because no port is selected
SPRC_E_NOT_MOD	Invalid operation because the port is not a modulation port
SPRC_OK	The modulation parameters have been read successfully

### Remarks

## SpRcClient::GetPayoutInfo

Get static payout information.

```
virtual SPRC_RESULT SpRcClient::GetPayoutInfo(  
    [out] SpRcPayoutInfo& PoInfo    // Static payout info  
);
```

### Parameters

*PoInfo*

Payout information, see Struct `SpRcPayoutInfo`.

### Result

SPRC_RESULT	Meaning
SPRC_E_COMMUNICATION	An error has occurred in the communication with the payout server
SPRC_E_NO_PORT	Invalid operation because no port is selected
SPRC_OK	The payout information has been read successfully

### Remarks

## SpRcClient::GetPayoutStatus

Get dynamic payout status.

```
virtual SPRC_RESULT SpRcClient::GetPayoutStatus(  
    [out] SpRcPayoutStatus& PoStatus // Dynamic payout status  
);
```

### Parameters

*PoInfo*

Payout information, see Struct `SpRcPayoutInfo`.

### Result

SPRC_RESULT	Meaning
SPRC_E_COMMUNICATION	An error has occurred in the communication with the payout server
SPRC_E_NO_PORT	Invalid operation because no port is selected
SPRC_OK	The payout status has been read successfully

### Remarks



## SpRcClient::GetRfPars

Get RF parameters.

```
virtual SPRC_RESULT SpRcClient::GetRfPars(  
    [out] SpRcRfPars& RfPars           // RF parameters  
);
```

### Parameters

*RfPars*

RF parameters, see Struct `SpRcRfPars`.

### Result

SPRC_RESULT	Meaning
SPRC_E_COMMUNICATION	An error has occurred in the communication with the playout server
SPRC_E_NO_PORT	Invalid operation because no port is selected
SPRC_E_NOT_MOD	Invalid operation because the port is not a modulation port
SPRC_OK	The RF parameters have been read successfully

### Remarks

## SpRcClient::GetSignalSource

Get current signal source (from file / test-generator).

```
virtual SPROC_RESULT SpRcClient::GetSignalSource(  
    [out] int& SignalSource // Signal source  
);
```

### Parameters

*SignalSource*

Current signal source.

Value	Meaning
SPRC_FROM_FILE	Data is read from file
SPRC_TEST_GENERATOR	The test-signal generator is used to generate an output signal

### Result

SPRC_RESULT	Meaning
SPRC_E_COMMUNICATION	An error has occurred in the communication with the playout server
SPRC_E_NO_PORT	Invalid operation because no port is selected
SPRC_OK	The signal source has been read successfully

### Remarks

## SpRcClient::GetSpiPars

Get DVB-SPI transmission parameters.

```
virtual SPRC_RESULT SpRcClient::GetSpiPars (  
    [out] SpRcAsiPars& SpiPars // DVB-SPI parameters  
);
```

### Parameters

*SpiPars*

DVB-SPI transmission parameters, see Struct **SpRcSpiPars**.

### Result

SPRC_RESULT	Meaning
SPRC_E_COMMUNICATION	An error has occurred in the communication with the playout server
SPRC_E_NO_PORT	Invalid operation because no port is selected
SPRC_E_NOT_SPI	Invalid operation because the port is not a DVB-SPI port
SPRC_OK	The DVB-SPI transmission parameters have been read successfully

### Remarks

## SpRcClient::GetTsgPars

Get test-signal generator parameters.

```
virtual SPRC_RESULT SpRcClient::GetTsgPars (  
    [out] SpRcTsgPars& TsgPars // Test-signal generator parameters  
);
```

### Parameters

*TsgPars*

Test-signal generator parameters, see Struct **SpRcTsgPars**.

### Result

SPRC_RESULT	Meaning
SPRC_E_COMMUNICATION	An error has occurred in the communication with the playout server
SPRC_E_NO_PORT	Invalid operation because no port is selected
SPRC_E_OP_NOT_SUPPORTED	Invalid operation because StreamXpress is not currently running in test-signal generator mode
SPRC_OK	The parameters have been read successfully

### Remarks

## SpRcClient::GetTsoipPars

Get TSolP transmission parameters.

```
virtual SPRC_RESULT SpRcClient::GetTsoipPars(  
    [out] SpRcTsoipPars& TsoipPars // TSoIP parameters  
);
```

### Parameters

*TsoipPars*

TSolP transmission parameters, see Struct **SpRcTsoipPars**.

### Result

SPRC_RESULT	Meaning
SPRC_E_COMMUNICATION	An error has occurred in the communication with the playout server
SPRC_E_NO_PORT	Invalid operation because no port is selected
SPRC_E_NOT_TSOIP	Invalid operation because the port is not a TSolP port
SPRC_OK	The TSolP transmission parameters have been read successfully

### Remarks

## SpRcClient::GetUseNit

Checking whether NIT is used for deriving parameters.

```
virtual SPRC_RESULT SpRcClient::GetUseNit(  
    [out] bool& UseNit  
);
```

### Parameters

*UseNit*

True if and only if StreamXpress will try to use NIT to set the modulation parameters.

### Result

SPRC_RESULT	Meaning
SPRC_E_COMMUNICATION	An error has occurred in the communication with the playout server
SPRC_E_NO_PORT	Invalid operation because no port is selected
SPRC_OK	The UseNit parameter has been read successfully

### Remarks

## SpRcClient::Normalise

Normalise the multipath channel modelling.

```
virtual SPRC_RESULT SpRcClient::Normalise();
```

### Parameters

### Result

SPRC_RESULT	Meaning
SPRC_E_NO_PORT	Invalid operation because no port is selected
SPRC_E_NOT_MOD	Invalid operation because the port is not a modulation port
SPRC_E_COMMUNICATION	An error has occurred in the communication with the playout server
SPRC_OK	The channel modelling settings have been normalised

### Remarks

## SpRcClient::OpenChannelModellingFile

Open file with settings for channel modelling.

```
virtual SPRC_RESULT SpRcClient::OpenChannelModellingFile (
    [in] std::wstring& Filename // Filename
);
```

### Parameters

*Filename*

Filename to be opened.

### Result

SPRC_RESULT	Meaning
SPRC_E_FILE_CANT_FIND	Can't find a file with the specified filename
SPRC_E_FILE_SYNTAX_ERROR	The specified file contains a syntax error
SPRC_E_COMMUNICATION	An error has occurred in the communication with the playout server
SPRC_E_NO_PORT	Invalid operation because no port is selected
SPRC_E_NOT_MOD	Invalid operation because the port is not a modulation port
SPRC_OK	The channel modelling settings file has been loaded successfully

### Remarks



## SpRcClient::OpenFile

Open file for playout.

```
virtual SPRC_RESULT SpRcClient::OpenFile(  
    [in] std::wstring&  Filename    // Filename  
);
```

### Parameters

*Filename*

Filename to be opened.

### Result

SPRC_RESULT	Meaning
SPRC_E_FILE_CANT_FIND	Can't find a file with the specified filename
SPRC_E_COMMUNICATION	An error has occurred in the communication with the playout server
SPRC_OK	The playout file has been opened successfully

### Remarks

## SpRcClient::SaveChannelModellingSettings

Save the channel modelling settings to a file.

```
virtual SPRC_RESULT SpRcClient::SaveChannelModellingSettings (  
    [in] std::wstring& Filename // Filename  
);
```

### Parameters

*Filename*

Filename to be opened.

### Result

SPRC_RESULT	Meaning
SPRC_E_FILE_CANT_CREATE	Error opening the file for writing
SPRC_E_COMMUNICATION	An error has occurred in the communication with the playout server
SPRC_E_NO_PORT	Invalid operation because no port is selected
SPRC_E_NOT_MOD	Invalid operation because the port is not a modulation port
SPRC_OK	The channel modelling settings have been saved successfully

### Remarks

## SpRcClient::SaveSettings

Save the settings to a file.

```
virtual SPRC_RESULT SpRcClient::SaveSettings(  
    [in] std::wstring& Filename // Filename  
);
```

### Parameters

*Filename*

Filename to be opened.

### Result

SPRC_RESULT	Meaning
SPRC_E_FILE_CANT_CREATE	Error opening the file for writing
SPRC_E_COMMUNICATION	An error has occurred in the communication with the playout server
SPRC_OK	The settings have been saved successfully

### Remarks

## SpRcClient::SetAsiPars

Set DVB-ASI transmission parameters.

```
virtual SPRC_RESULT SpRcClient::SetAsiPars (
    [in] SpRcAsiPars AsiPars // ASI parameters
);
```

### Parameters

*AsiPars*

DVB-ASI transmission parameters, see **Struct SpRcAsiPars**.

### Result

SPRC_RESULT	Meaning
SPRC_E_COMMUNICATION	An error has occurred in the communication with the playout server
SPRC_E_NO_PORT	Invalid operation because no port is selected
SPRC_E_NOT_ASI	Invalid operation because the port is not an ASI port or is operating in SDI mode
SPRC_E_POLARITY	The polarity specified in <i>AsiPars</i> is not supported by the device
SPRC_E_TXMODE	The transmit mode specified in <i>AsiPars</i> is not compatible with the Transport-Stream file
SPRC_OK	The DVB-ASI transmission parameters have been set successfully

### Remarks

## SpRcClient::SetChannelModellingPars

Set Channel Modelling parameters.

```
virtual SPRC_RESULT SpRcClient::SetChannelModellingPars (  
    [in] SpRcCmPars CmPars           // Channel Modelling parameters  
);
```

### Parameters

*CmPars*

Channel Modelling parameters, see **Struct SpRcCmPars**.

### Result

SPRC_RESULT	Meaning
SPRC_E_COMMUNICATION	An error has occurred in the communication with the playout server
SPRC_E_NO_PORT	Invalid operation because no port is selected
SPRC_E_NOT_MOD	Invalid operation because the port is not a modulation port
SPRC_OK	The Channel Modelling parameters have been set successfully

### Remarks

## SpRcClient::SetCmmbPars

Set CMMB parameters.

```
virtual SPRC_RESULT SpRcClient::SetCmmbPars (  
    [in] SpRcCmmbPars CmmbPars // CMMB parameters  
);
```

### Parameters

*CmmbPars*

CMMB parameters, see **Struct SpRcCmmbPars**.

### Result

SPRC_RESULT	Meaning
SPRC_E_COMMUNICATION	An error has occurred in the communication with the playout server
SPRC_E_NO_PORT	Invalid operation because no port is selected
SPRC_E_NOT_MOD	Invalid operation because the port is not a modulation port
SPRC_E_NOT_CMMB	Invalid operation because CMMB modulation is not configured
SPRC_OK	The parameters have been set successfully

### Remarks

## SpRcClient::SetDvbT2Group

Set DVB-T2 standard parameter set; e.g. 'V125' parameters.

```
virtual SPRC_RESULT SpRcClient::SetDvbT2Group(  
    [in] SpRcDvbT2Group DvbT2Group // DVB-T2 parameter set  
);
```

### Parameters

*DvbT2Group*

DVB-T2 standard parameter set, see Struct `SpRcDvbT2Group`.

### Result

SPRC_RESULT	Meaning
SPRC_E_COMMUNICATION	An error has occurred in the communication with the playout server
SPRC_E_NO_PORT	Invalid operation because no port is selected
SPRC_E_NOT_MOD	Invalid operation because the port is not a modulation port
SPRC_E_NOT_DVB-T2	Invalid operation because DVB-T2 modulation is not configured
SPRC_OK	The parameters have been set successfully

### Remarks

## SpRcClient::SetDvbT2Pars

Set DVB-T2 parameters.

```
virtual SPRC_RESULT SpRcClient::SetDvbT2Pars (  
    [in] SpRcDvbT2Pars DvbT2Pars // DVB-T2 parameters  
);
```

### Parameters

*DvbT2Pars*

DVB-T2 parameters, see **Struct SpRcDvbT2Pars**.

### Result

SPRC_RESULT	Meaning
SPRC_E_COMMUNICATION	An error has occurred in the communication with the playout server
SPRC_E_NO_PORT	Invalid operation because no port is selected
SPRC_E_NOT_MOD	Invalid operation because the port is not a modulation port
SPRC_E_NOT_DVB-T2	Invalid operation because DVB-T2 modulation is not configured
SPRC_OK	The parameters have been set successfully

### Remarks



## SpRcClient::SetHwNoisePars

Set Noise parameters for modulators DTA-107 and DTA-2107.

```
virtual SPRC_RESULT SpRcClient::SetHwNoisePars (  
    [in] SpRcHwNoisePars HwNoisePars // Noise parameters  
);
```

### Parameters

*HwNoisePars*

Noise parameters, see Struct `SpRcHwNoisePars`.

### Result

SPRC_RESULT	Meaning
SPRC_E_COMMUNICATION	An error has occurred in the communication with the playout server
SPRC_E_NO_PORT	Invalid operation because no port is selected
SPRC_E_NOT_MOD	Invalid operation because the port is not a modulation port
SPRC_OK	The parameters have been set successfully

### Remarks

## SpRcClient::SetIsdbtPars

Set ISDB-T modulation parameters.

```
virtual SPRC_RESULT SpRcClient::SetIsdbtPars (  
    [in] SpRcIsdbtPars IsdbtPars    // ISDB-T parameters  
);
```

### Parameters

*IsdbtPars*

ISDB-T modulation parameters, see **Struct SpRcIsdbtPars**.

### Result

SPRC_RESULT	Meaning
SPRC_E_COMMUNICATION	An error has occurred in the communication with the playout server
SPRC_E_NO_PORT	Invalid operation because no port is selected
SPRC_E_NOT_ISDBT	Invalid operation because the port is operating in ISDB-T modulation mode
SPRC_E_NOT_MOD	Invalid operation because the port is not a modulation port
SPRC_OK	The modulation parameters have been set successfully

### Remarks

## SpRcClient::SetIqGain

Set IQ gain parameter.

```
virtual SPROC_RESULT SpRcClient::SetIqGain(  
    [in] int IqGain // IQ gain  
);
```

### Parameters

*IqGain*

The gain of the IQ signal.

### Result

SPRC_RESULT	Meaning
SPRC_E_COMMUNICATION	An error has occurred in the communication with the playout server
SPRC_E_NO_PORT	Invalid operation because no port is selected
SPRC_E_NOT_MOD	Invalid operation because the port is not a modulation port
SPRC_OK	The modulation parameters have been set successfully

### Remarks

## SpRcClient::SetLoopFlags

Set loop-adaptation flags.

```
virtual SPRC_RESULT SpRcClient::SetLoopFlags(
    [in] int LoopFlags // Loop-adaptation flags
);
```

### Parameters

*LoopFlags*

Loop-adaptation flags encoded in flags that may be OR-ed together.

Value	Meaning
SPRC_LOOP_CC	Adapt continuity counters
SPRC_LOOP_PCR	Adapt PCR
SPRC_LOOP_TDT	Adapt TDT
SPRC_LOOP_WRAP	Auto wrap-around

### Result

SPRC_RESULT	Meaning
SPRC_E_COMMUNICATION	An error has occurred in the communication with the playout server
SPRC_E_NO_PORT	Invalid operation because no port is selected
SPRC_OK	The loop-adaptation flags have been set successfully

### Remarks

## SpRcClient::SetModPars

Set modulation parameters for all modulation standards except ISDB-T and DVB-T2.

```
virtual SPRC_RESULT SpRcClient::SetModPars(  
    [in] SpRcModPars ModPars // Modulation parameters  
);
```

### Parameters

*ModPars*

Modulation parameters, see **Struct SpRcModPars**.

### Result

SPRC_RESULT	Meaning
SPRC_E_COMMUNICATION	An error has occurred in the communication with the playout server
SPRC_E_NO_PORT	Invalid operation because no port is selected
SPRC_E_NOT_MOD	Invalid operation because the port is not a modulation port
SPRC_OK	The modulation parameters have been set successfully

### Remarks

## SpRcClient::SetPlayoutState

Set playout state (play/pause/stop).

```
virtual SPRC_RESULT SpRcClient::SetPlayoutState(
    [in] int PlayoutState // Playout state
);
```

### Parameters

*PlayoutState*

New playout state.

Value	Meaning
SPRC_STATE_PAUSE	Pause
SPRC_STATE_PLAY	Play
SPRC_STATE_STOP	Stop

### Result

SPRC_RESULT	Meaning
SPRC_E_COMMUNICATION	An error has occurred in the communication with the playout server
SPRC_E_INV_STATE	Invalid playout state
SPRC_E_NO_PORT	Invalid operation because no port is selected
SPRC_OK	The playout state has been set successfully

### Remarks

## SpRcClient::SetRfPars

Set RF parameters.

```
virtual SPROC_RESULT SpRcClient::SetRfPars(  
    [in] SpRcRfPars  RfPars           // RF parameters  
);
```

### Parameters

*RfPars*

RF parameters, see Struct `SpRcPars`.

### Result

SPRC_RESULT	Meaning
SPRC_E_COMMUNICATION	An error has occurred in the communication with the playout server
SPRC_E_INV_FREQ	The centre frequency in <i>RfPars</i> is not supported by the upconverter on this modulator
SPRC_E_INV_LEVEL	The level in <i>RfPars</i> is not supported by this modulator
SPRC_E_NO_PORT	Invalid operation because no port is selected
SPRC_E_NOT_MOD	Invalid operation because the port is not a modulation port
SPRC_OK	The RF parameters have been set successfully

### Remarks

## SpRcClient::SetSignalSource

Change signal source (from file / test-generator).

```
virtual SPRC_RESULT SpRcClient::SetSignalSource(
    [in] int  SignalSource      // Signal source
);
```

### Parameters

*SignalSource*

New signal source.

Value	Meaning
SPRC_FROM_FILE	Data is read from file
SPRC_TEST_GENERATOR	The test-signal generator is used to generate an output signal

### Result

SPRC_RESULT	Meaning
SPRC_E_COMMUNICATION	An error has occurred in the communication with the playout server
SPRC_E_OP_NOT_SUPPORTED	Current modulation type is not supported
SPRC_E_NO_PORT	Invalid operation because no port is selected
SPRC_OK	The signal source has been changed successfully

### Remarks



## SpRcClient::SetSpiPars

Set DVB-SPI transmission parameters.

```
virtual SPRC_RESULT SpRcClient::SetAsiPars(
    [in] SpRcSpiPars SpiPars // DVB-SPI parameters
);
```

### Parameters

*SpiPars*

DVB-SPI transmission parameters, see **Struct SpRcSpiPars**.

### Result

SPRC_RESULT	Meaning
SPRC_E_COMMUNICATION	An error has occurred in the communication with the playout server
SPRC_E_NO_PORT	Invalid operation because no port is selected
SPRC_E_NOT_SPI	Invalid operation because the port is not a DVB-SPI port
SPRC_E_TXMODE	The transmit mode specified in <i>SpiPars</i> is not compatible with the Transport-Stream file
SPRC_OK	The DVB-SPI transmission parameters have been set successfully

### Remarks

## SpRcClient::SetTsgPars

Set test-signal generation parameters.

```
virtual SPRC_RESULT SpRcClient::SetTsoipPars(  
    [in] SpRcTsgPars TsgPars // test-signal generation parameters  
);
```

### Parameters

*TsgPars*

Test-signal generation parameters, see Struct `SpRcTsgPars`.

### Result

SPRC_RESULT	Meaning
SPRC_E_COMMUNICATION	An error has occurred in the communication with the playout server
SPRC_E_NO_PORT	Invalid operation because no port is selected
SPRC_E_OP_NOT_SUPPORTED	Invalid operation because StreamXpress is not currently running in test-signal generator mode
SPRC_OK	The parameters have been set successfully

### Remarks

## SpRcClient::SetTsoipPars

Set TSolP transmission parameters.

```
virtual SPRC_RESULT SpRcClient::SetTsoipPars (  
    [in] SpRcTsoipPars  TsoipPars    // TSoIP parameters  
);
```

### Parameters

*TsoipPars*

TSolP transmission parameters, see Struct **SpRcTsoipPars**.

### Result

SPRC_RESULT	Meaning
SPRC_E_COMMUNICATION	An error has occurred in the communication with the playout server
SPRC_E_NO_PORT	Invalid operation because no port is selected
SPRC_E_NOT_TSOIP	Invalid operation because the port is not a TSolP port
SPRC_OK	The TSolP transmission parameters have been set successfully

### Remarks

## SpRcClient::SetTsRate

Set Transport-Stream rate

```
virtual SPRC_RESULT SpRcClient::SetTsRate(  
    [in] Int    TsRate           // Transport-Stream rate  
);
```

### Parameters

*TsRate*

Transport-Stream rate in bits per second; 188 bytes per Transport Packets

### Result

SPRC_RESULT	Meaning
SPRC_E_COMMUNICATION	An error has occurred in the communication with the playout server
SPRC_E_NO_PORT	Invalid operation because no port is selected
SPRC_OK	The rate has been set successfully

### Remarks

## SpRcClient::SetUseNit

Enable or disable the use of NIT for deriving modulation parameters

```
virtual SPRC_RESULT SpRcClient::SetUseNit(  
    [in] Bool UseNit  
);
```

### Parameters

*UseNit*

Whether or not NIT should be used.

### Result

SPRC_RESULT	Meaning
SPRC_E_COMMUNICATION	An error has occurred in the communication with the playout server
SPRC_E_NO_PORT	Invalid operation because no port is selected
SPRC_OK	The parameter has been set successfully

### Remarks

## SpRcClient::WaitForCondition

Wait for a certain condition.

```
virtual SPRC_RESULT SpRcClient::WaitForCondition(
    [in] int Condition,          // Playout state to wait for
    [in] int TimeOut           // Maximum time to wait in ms
);
```

### Parameters

*Condition*

Condition to wait for.

Value	Meaning
SPRC_COND_STOPPED	Playout server is in a stopped state; In this context, pause is not considered a stopped state

*TimeOut*

Time-out period in ms. If this parameter is -1 no time out is applied.

### Result

SPRC_RESULT	Meaning
SPRC_E_COMMUNICATION	An error has occurred in the communication with the playout server
SPRC_E_INV_CONDITION	An invalid condition has been specified
SPRC_OK	The condition has occurred
SPRC_TIME_OUT	The time out has triggered

### Remarks