

# How to use the R&S Spectrum Analyzer IVI-COM Driver

## Contents

1	Overview .....	2
2	Additional Help .....	2
3	Using the IVI-COM Driver in Microsoft Visual Basic .....	3
3.1	Adding a reference to the IVI COM driver objects .....	3
3.2	Creating an instance of the IVI COM driver object .....	3
3.3	Using the Driver .....	4
4	Using the IVI-COM Driver in C++ .....	5
4.1	Importing the IVI COM driver interfaces .....	5
4.2	Initializing COM .....	5
4.3	Using the Driver .....	5
4.4	Error Handling .....	6
5	Using the IVI-COM Driver in Scripting Environments .....	7
5.1	Creating the script file .....	7
5.2	Executing the script file .....	7
5.3	Using the class driver interface .....	7
6	Optional and Advanced Features .....	8
6.1	Hardware Simulation .....	8
6.2	Range Checking and Coercion Recording .....	8
6.3	The IVI Configuration Store .....	8
6.4	The IVI Class Factory .....	9

# 1 Overview

This document describes how to use the R&S Spectrum Analyzer IVI COM driver in various programming environments. The same information is also available in the online documentation, which, in addition, also contains a detailed description of all the instrument specific functions and the IviSpecAn class driver functions.

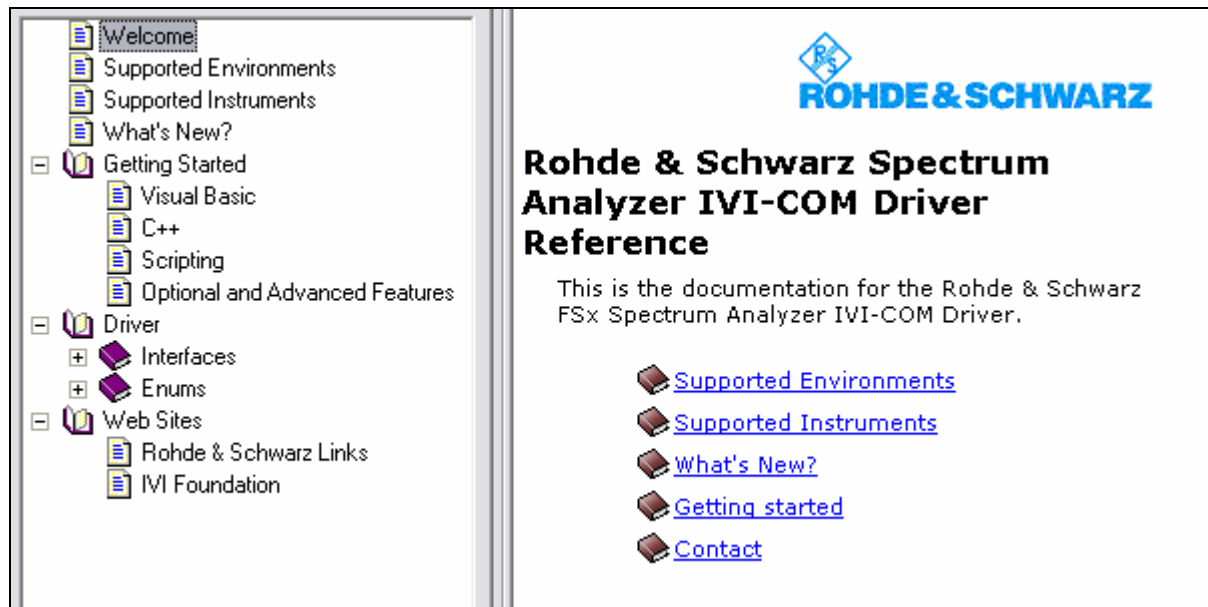
As far as the RsFs IVI COM driver itself is concerned it is useful to know that the driver is an IVI Class-compliant Specific Driver, providing an IviSpecAn class driver interface for the spectrum analyzer.

For general information about the IVI terms and features please refer to the original IVI documentation which can be found at [www.ivifoundation.org](http://www.ivifoundation.org).

Technical data of the driver may be found in the Readme file.

# 2 Additional Help

In addition, extra documentation is also included in compressed HTML format (Windows CHM help file) and stored together with the driver in the **c:\Programs\IVI\RsFs** directory.



### 3 Using the IVI-COM Driver in Microsoft Visual Basic

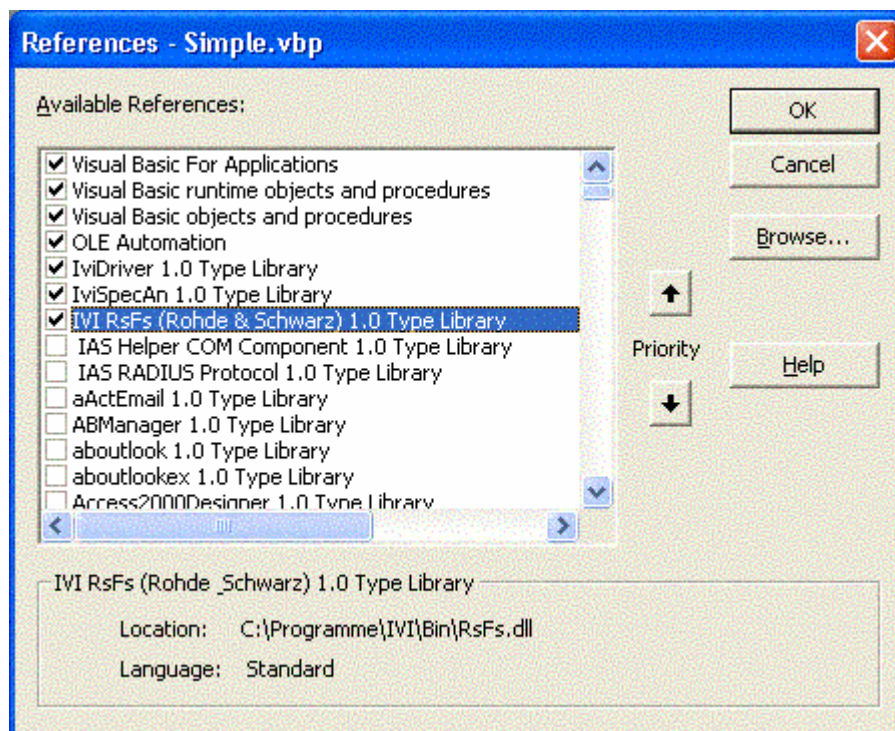
This chapter describes step by step how to use the IVI driver in Visual Basic 6 (VB6).

#### 3.1 Adding a reference to the IVI COM driver objects

Create a new Visual Basic project.

Select the Visual Basic menu item "Project/References".

In the list box, locate the line "IVI RsFs (Rohde & Schwarz) X.X Type Library" denoting the R&S FSx IVI driver for the instrument, set the check mark in the checkbox.



If you intend to use the *IviSpecAn* class compliant interfaces, also check the line "IviSpecAn X.X Type Library".

#### 3.2 Creating an instance of the IVI COM driver object

At the top of the Visual Basic source code window type in

```
Dim mySpecAn As New
```

After entering a space after "New" there appears a drop down listbox offering the available data types. Select the data type "RsFs". The methods and properties of the FSx specific driver are now accessible via the object variable "mySpecAn". (There are more portable methods to create a driver object instance, see below).

### 3.3 Using the Driver

To initialize the driver, the function "mySpecAn.Initialize()" is to be called. This may be done in the Visual-Basic function Form\_Load() or another appropriate place in the source code.

In the Visual Basic source code window type in "Call mySpecAn.". In the drop-down box locate the "Initialize" function and press the Tab key. Then enter the opening bracket '(' for the argument list.

The first argument of `Initialize()` is the VISA resource descriptor, a string like "GPIB0::20::INSTR". The `IdQuery` and `Reset` arguments may both be set to "False", the option string may be empty. The source code for a minimal test program looks like this:

```
Call mySpecAn.Initialize("GPIB0::20::INSTR", False, False, "")
Dim bSuccess As Boolean
bSuccess = mySpecAn.Initialized
If bSuccess = False Then
MsgBox "Failed to initialize", vbCritical, "Error"
End If

' Set some instrument parameters - here center frequency and span
Call mySpecAn.Frequency.ConfigureCenterSpan(128000000#, 10000000#)
'...

' When done, close the driver session
Call mySpecAn.Close
```

## 4 Using the IVI-COM Driver in C++

This chapter describes step by step how to use the IVI driver in C++.

### 4.1 Importing the IVI COM driver interfaces

Add the following lines at the top of the CPP file:

```
#define _WIN32_DCOM
#include <objbase.h>
#include <atlbase.h>

#import "IviDriverTypeLib.dll"
#import "IviSpecAnTypeLib.dll"
#import "RsFs.dll"

using namespace RsFs_RohdeAndSchwarzLib;

// forward declarations
static void run(void);
```

### 4.2 Initializing COM

To enable and disable COM requires calls to the windows API functions [CoInitializeEx\(\)](#) and [CoUninitialize\(\)](#). Sample code for this is

```
int main()
{
    HRESULT hr(CoInitializeEx(0, COINIT_MULTITHREADED));
    run();
    CoUninitialize();
    return 0;
}
```

### 4.3 Using the Driver

After initializing COM the drivers functions may be called.  
A simple test is:

```
static void run()
{
    IRsFsPtr mySpecAnPtr;
    HRESULT hr(mySpecAnPtr.CreateInstance(__uuidof(RsFs)));
    mySpecAnPtr->Initialize("GPIB0::20::INSTR", VARIANT_FALSE,
                           VARIANT_FALSE, "");
    if (mySpecAnPtr->GetInitialized())
    {
        IRsFsFrequencyPtr myFrequencyPtr = mySpecAnPtr->GetFrequency();
        myFrequencyPtr->put_Start(120e6);
    }
    mySpecAnPtr->Close();
}
```

The essential steps are to retrieve a pointer `mySpecAn` to a driver object, and then to `Initialize()` and finally `Close()` the driver.

The first argument of the `Initialize()` function is the VISA resource descriptor, a string like "GPIB0::20::INSTR". The `IdQuery` and `Reset` arguments may both be set to `VARIANT_FALSE`, the `Option String` may be empty.

#### 4.4 Error Handling

The recommended error handling method when using `#import` is to overwrite the `_com_raise_error()` function (see the MSDN under this function name). An example is:

```
void __stdcall _com_raise_error(HRESULT hr, IErrorInfo* pEI)
{
    CComBSTR Descr;
    pEI->GetDescription(&Descr);
    _bstr_t Text(Descr);
    printf("HRESULT = %0X\r\n%s\r\n", hr, static_cast<const char*>(Text));
    exit(1);
}
```

## 5 Using the IVI-COM Driver in Scripting Environments

The IVI COM driver may also be used in MS Windows Scripting Host VBS (Visual Basic Script) or JS (Java Script) scripts.

This is a quick and simple way to control the instrument. The disadvantages are that functions with array arguments cannot be accessed, and that normal text editors do not support intellisense. (A script debugger can be downloaded from the Microsoft Scripting web page).

This page describes step by step how to use the IVI driver in a Visual Basic Script.

### 5.1 Creating the script file

Use Notepad or another ASCII editor to create a new file, add the lines

```
Set mySpecAn = CreateObject("RsFs.RsFs.1")
Call mySpecAn.Initialize("GPIB0::20::INSTR", False, False, "")
If mySpecAn.Initialized Then
    MsgBox "Success"
    ' Get some instrument parameters - here stop frequency
    MsgBox "mySpecAn.Frequency.Stop = " & mySpecAn.Frequency.Stop
Else
    MsgBox "Error"
End If
Call mySpecAn.Close
```

The argument string for CreateObject is the ProgId of the driver. The first argument for [Initialize\(\)](#) is the Visa resource descriptor, an example is "GPIB0::28::INSTR".

Save the file under the name *RsFs.vbs*.

### 5.2 Executing the script file

To execute the script, enter the file name at the command prompt or double-click on the file symbol in the explorer.

### 5.3 Using the class driver interface

To exclusively use the class compliant *IIviSpecAn* interface for accessing the hardware, first create a FSx specific driver object as described above, then extract a class compliant interface object,

```
Dim ClsSpecAn
Set ClsSpecAn = mySpecAn.IviSpecAn
```

The class compliant functions for the FSx can now be accessed via the *IIviSpecAn* object *ClsSpecAn*.

## 6 Optional and Advanced Features

The IVI driver architecture contains a lot of generic functionality.

The IVI "*Inherent Capabilities*" are available under the [RootInterface](#), or under the [IIviDriverIdentity](#), [IIviDriverOperation](#) and [IIviDriverUtility](#) interfaces.

Some features are described below. For a detailed description please refer to the function descriptions in this online help or to the original [IVI documentation](#).

### 6.1 Hardware Simulation

Simulation of the hardware can be enabled by setting a flag in the last argument of the [Initialize\(\)](#) function (the *OptionString*).

```
[Visual Basic and VB Script]
Call Obj.Initialize("GPIB0::20::INSTR", False, False, "Simulate=True")
```

```
[C++]
Obj->Initialize("GPIB0::20::INSTR", VARIANT_FALSE, VARIANT_FALSE, "Simulate=True")
```

With an *OptionString* "Simulate=True" there won't be any accesses to the hardware. Range checks are still performed as far as possible, and the driver will return simulated parameter values.

### 6.2 Range Checking and Coercion Recording

*Range Checking* and *Coercion Recording* may also be enabled in the *OptionString*. An *OptionString* that enables both features reads

"RangeCheck=True,RecordCoercions=True".

An IVI driver is not required to repeat all the range checks done in the instrument.

If the driver coerces a value to the valid range and coercion recording is enabled, it will generate a coercion record. These records can be read with the [IIviDriverOperation](#) function [GetNextCoercionRecord\(\)](#).

### 6.3 The IVI Configuration Store

The first argument of the [Initialize\(\)](#) function need not be a VISA resource string. Instead it may also be an *IVI Logical Name* identifying an *IVI Driver Session* in the *IVI Configuration Store* (the central IVI data repository, an XML file).

Every *IVI Driver Session* record in the Configuration Store contains values for the VISA resource string and the other [Initialize\(\)](#) arguments.

If the first argument of the [Initialize\(\)](#) function specifies an *IVI Driver Session*, then the arguments for the function are taken from the corresponding *IVI Driver Session* record in the Configuration Store. The purpose of this construct is to reconfigure the application without touching the source code of the application. Using the IVI Configuration Store requires that the *IVI Shared Components* be installed.



## 6.4 The IVI Class Factory

Another step to make the application code more generic would be to use the *IVI Class Factory*. The purpose of the *IVI Class Factory* is to create a driver object from an *IVI Driver Session* name. Assume that the *IVI Configuration Store* contains a *IVI Driver Session* record with logical name "RsFs.DriverSession". Then the following code is an alternative to creating the object directly (as described above):

```
[Visual Basic]
Dim Factory As New IviSessionFactory
Dim Obj2 As RsFs
Set Obj2 = Factory.CreateDriver("RsFs.DriverSession")
```

```
[C++]
#import "IviSessionFactory.dll"
using namespace IVICESSIONFACTORYLib;
IIviSessionFactoryPtr ObjFac;
HRESULT hr(ObjFac.CreateInstance(__uuidof(IviSessionFactory)));
IRsFsPtr Obj = ObjFac->CreateDriver("RsFs.DriverSession");
```

(Using the session factory with Visual Basic requires a reference to *"IviSessionFactory X.X Type Library"* under the menu item /Project/References).

A more useful application of the *IVI Class Factory* would be to use it for creating an *IIviSpecAn* class driver object:

```
Dim Factory As New IviSessionFactory
Dim ClsObj2 As IIviSpecAn
Set ClsObj2 = Factory.CreateDriver("RsFs.DriverSession")
```