How to use the SMU IVI-COM driver

Contents

1 Overview 2
2 Additional Help 2
3 Using the IVI-COM Driver in Microsoft Visual Basic 2
  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 3
  3.4 Using the class driver interface 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 with Ansi C 7
  5.1 Importing the IVI COM driver interfaces 7
  5.2 Initializing COM 7
  5.3 Using the Driver 7
  5.4 Using the class driver interface 8
  5.5 Error Codes 8
6 Using the IVI-COM Driver in Scripting Environments 9
  6.1 Creating the script file 9
  6.2 Executing the script file 9
  6.3 Using the class driver interface 9
7 Optional and Advanced Features 10
  7.1 Hardware Simulation 10
  7.2 Range Checking and Coercion Recording 10
  7.3 The IVI Configuration Store 10
  7.4 The IVI Class Factory 11
  7.5 The IVI Event Server 11

1 Overview

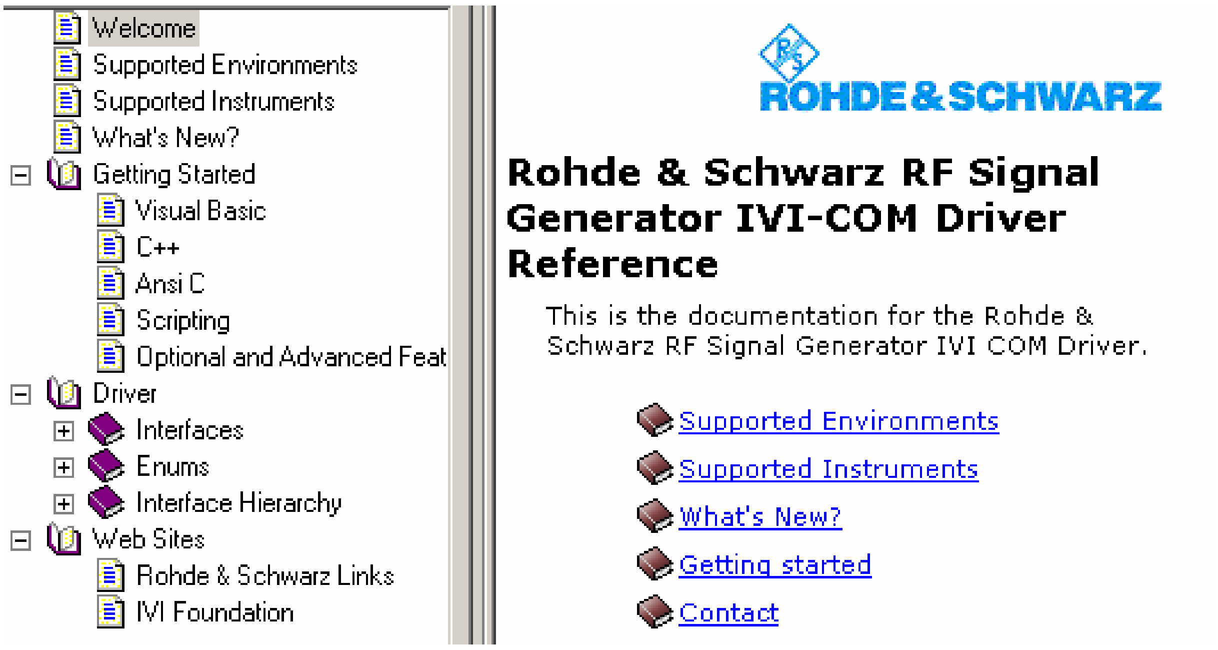
This document describes how to use the RsSMU IVI COM driver in various programming environments.
(The same information is also available in the online documentation, which, in addition, also contains a detailed description of all the instrument specific functions and the IviRFSigGen class driver functions).

As far as the RsSMU IVI COM driver itself is concerned it is useful to know that the driver is an IVI Class-compliant Specific Driver, providing an IviRFSigGen class driver interface for each signal generator path available in the instrument (mostly 1 or 2, path 'A' and path 'B').
To use class driver functions a path specific IviRFSigGen interface must be extracted from the driver. (The class driver functions only cover a small fraction of the complete set of instrument specific functions).

For general information about the IVI terms and features please use the original IVI documentation, www.ivifoundation.org.
Technical data of the driver may be found in the Readme file.

2 Additional Help

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



Rohde & Schwarz RF Signal Generator IVI-COM Driver Reference

This is the documentation for the Rohde & Schwarz RF Signal Generator IVI COM Driver.

Supported Environments

Supported Instruments

What's New?

Getting started

Contact

3 Using the IVI-COM Driver in Microsoft Visual Basic

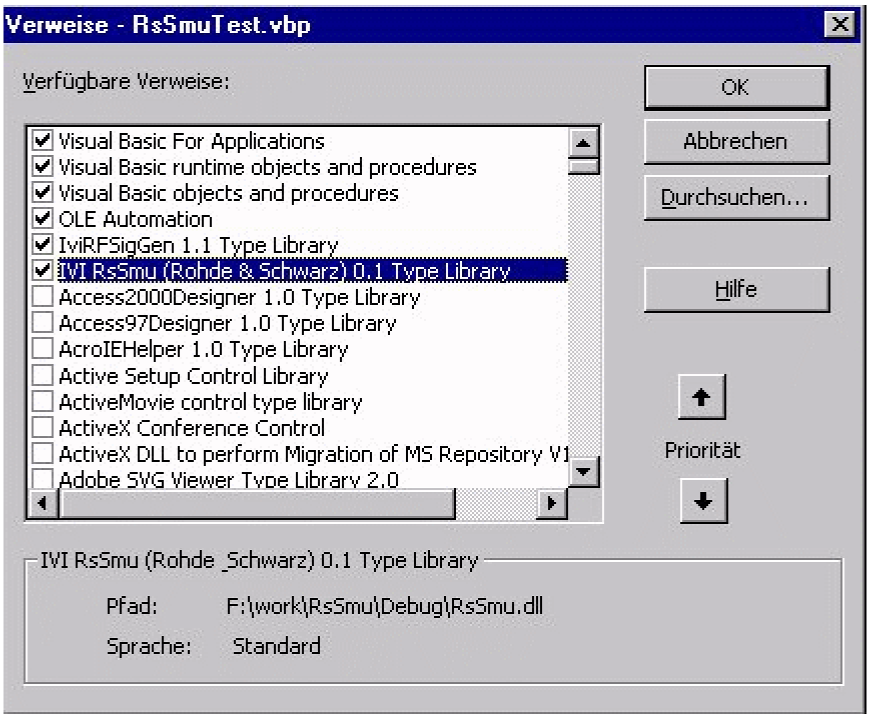
This chapter describes step by step how to use the IVI driver in Visual Basic.

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 RsSmu (Rohde & Schwarz) X.X Type Library" denoting the R&S SMU IVI driver for the instrument, set the check mark in the checkbox.



- [x] Visual Basic For Applications
- [x] Visual Basic runtime objects and procedures
- [x] Visual Basic objects and procedures
- [x] OLE Automation
- [x] IviRFSigGen 1.1 Type Library
- [x] IVI RsSmu (Rohde & Schwarz) 0.1 Type Library
- [ ] Access2000Designer 1.0 Type Library
- [ ] Access97Designer 1.0 Type Library
- [ ] AcroIEHelper 1.0 Type Library
- [ ] Active Setup Control Library
- [ ] ActiveMovie control type library
- [ ] ActiveX Conference Control
- [ ] ActiveX DLL to perform Migration of MS Repository V1
- [ ] Adobe SVG Viewer Type Library 2.0

IVI RsSmu (Rohde _Schwarz) 0.1 Type Library

Pfad: F:\work\RsSmu\Debug\RsSmu.dll

Sprache: Standard

If you intend to use the *IviRFSigGen* class compliant interfaces, also check the line "IviRFSigGen 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 Obj As New
```

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

3.3 Using the Driver

To initialize the driver, the function "Obj.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 Obj.". In the drop-down box select 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::28::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 Obj.Initialize("GPIB0::28::INSTR", False, False, "")
Dim bSuccess As Boolean
bSuccess = Obj.Initialized
Obj.Paths.Item(1).RF.Frequency = 2.2E7
...
Call Obj.Close
```

3.4 Using the class driver interface

To access the instrument via the class compliant IIviRFSigGen interface, first create a R&S SMU specific driver object as described above, then extract a class compliant interface,

```
  Dim ClsObj As IIviRFSigGen
  Set ClsObj = Obj.Paths.Item(1)
```

The class compliant functions for the R&S SMU path 1 are now available via the IIviRFSigGen interface object ClsObj.

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 "IviRFSigGenTypeLib.dll"
#import "RsSmu.dll"

using namespace IviRFSigGenLib;
using namespace RsSmu_Lib;
```

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:

```
void run()
{
   IRsSmuPtr Obj;
   HRESULT hr(Obj.CreateInstance(__uuidof(RsSmu)));
   Obj->Initialize("GPIB0::28::INSTR", VARIANT_FALSE, VARIANT_FALSE, "");
   if (Obj->GetInitialized())
   {
      ISmuPathPtr Path1 = Obj->Paths->GetItem("1");
      Path1->RF->PutFrequencyMode(SmuRfFrequencyModeFixed);
      Path1->RF->PutFrequency(1.4E6);
   }
   Obj->Close();
}
```

The essential steps are to retrieve a pointer `Obj` 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 with Ansi C

This page describes step by step how to use the IVI driver in an Ansi C program.

5.1 Importing the IVI COM driver interfaces

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

```
#define _WIN32_DCOM
#include <objbase.h>
#define COBJMACROS
#define CINTERFACE
#include "IviDriverTypeLib.h"
#include "IviRFSigGenTypeLib.h"
#include "RsSmu.h"
#include "RsSmu_i.c"
```

You will have add the compiler include path to the IVI specific header files, normally this is `"c:/Programs/Ivi/Include"`.

5.2 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);
   if (FAILED(hr))
   {
      return -1;
   }
   run();
   CoUninitialize();
   return 0;
}
```

5.3 Using the Driver

After initializing COM the driver can be used. A simple test is:

```
void run()
{
   // Create the RsSmu COM object and get an interface pointer
   IRsSmu* Obj;
   HRESULT hr = CoCreateInstance(CLSID_RsSmu, NULL, CLSCTX_SERVER, IID_IRsSmu,
(void**)&Obj);
   if (FAILED(hr))
   {
      return;
   }
   // Initialize the driver
   BSTR ResourceDescr = SysAllocString(OLESTR("GPIB0::28::INSTR"));
   BSTR OptionStr = SysAllocString(OLESTR("Simulate=False"));
   hr = IRsSmu_Initialize(Obj, ResourceDescr, VARIANT_TRUE, VARIANT_TRUE, OptionStr);
   if (FAILED(hr))
   {
      return;
   }
   SysFreeString(ResourceDescr);
   SysFreeString(OptionStr);
   // Get and use other interfaces
```

```
    ISmuReferenceOscillator* RefOsc;
    hr = IRsSmu_get_ReferenceOscillator(Obj, &RefOsc);
    SmuReferenceOscillatorExtFrequEnum Freq;
    hr = ISmuReferenceOscillator_get_ExternalFrequency(RefOsc, &Freq);
    IRsSmu_Close(Obj);
    // Release the interfaces
    ISmuReferenceOscillator_Release(RefOsc);
    IRsSmu_Release(Obj);
}
```

The essential steps are to get an interface pointer Obj 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::28::INSTR". The *IdQuery* and *Reset* arguments may both be set to *VARIANT_FALSE*, the *Option String* may be empty.

5.4 Using the class driver interface

To access the instrument via the class compliant IIviRFSigGen interface, first create a SMU specific driver object as described above, then extract a class compliant interface,

```
ISmuPaths* Paths;
HRESULT hr = IRsSmu_get_Paths(Obj, &Paths);
ISmuPath* Path;
BSTR Select = SysAllocString(OLESTR("1"));
ISmuPaths_get_Item(Paths, Select, &Path);
SysFreeString(Select);
IIviRFSigGen* ClsObj;
ISmuPath_get_ClassDriver(Path, &ClsObj);
IIviRFSigGen_Close(ClsObj);
// release interfaces...
```

The class compliant functions for the SMU path 1 are now available via the IIviRFSigGen interface pointer ClsObj.

5.5 Error Codes

The HRESULT error codes may be generic COM error codes, inherent IVI driver error codes, RFSigGen class driver error codes, or SMU specific error codes.
Hint: Use "decode.exe" to decode generic COM errors.

6 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.

6.1 Creating the script file

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

```
Set Obj = CreateObject("RsSmu.RsSmu.1")
Call Obj.Initialize("GPIB0::28::INSTR", False, False, "")
If Obj.Initialized Then
   MsgBox "Success"
   MsgBox "LfGenerator.Frequency = " & Obj.LfGenerator.Frequency
Else
   MsgBox "Error"
End If
Call Obj.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 *RsSmu.vbs*.

6.2 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.

6.3 Using the class driver interface

To use the class compliant IIviRFSigGen interface for accessing the hardware, first create a R&S SMU specific driver object as described above, then extract a class compliant interface object,

```
Dim ClsObj
Set ClsObj = Obj.Paths.Item(1).ClassDriver
```

The class compliant functions for the R&S SMU path 1 are now available via the IIviRFSigGen object ClsObj.

7 Optional and Advanced Features

The IVI driver architecture contains a lot of generic functionality.
The IVI *"Inherent Capabilities"* are available under the Root Interface, 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.

7.1 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::28::INSTR", False, False, "Simulate=True")
```

```
[C++]
Obj->Initialize("GPIB0::28::INSTR", VARIANT_FALSE, VARIANT_FALSE, "Simulate=True")
```

```
[Ansi C]
BSTR ResourceDescr = SysAllocString(OLESTR("GPIB0::28::INSTR"));
BSTR OptionStr = SysAllocString(OLESTR("Simulate=True"));
IRsSmu_Initialize(Obj, ResourceDescr, VARIANT_FALSE, VARIANT_FALSE, OptionStr);
```

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.

7.2 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().

7.3 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.

7.4 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 an *IVI Driver Session* record with logical name "RsSmu.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 RsSmu
Set Obj2 = Factory.CreateDriver("RsSmu.DriverSession")
```

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

(Using the session factory with Visual Basic requires a reference to *"IviSessioFactory 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 *IIviRFSigGen* class driver object:

```
Dim Factory As New IviSessionFactory
Dim ClsObj2 As IIviRFSigGen
Set ClsObj2 = Factory.CreateDriver("RsSmu.DriverSession")
```

However, this does not work for the R&S SMU: The SMU contains a collection of paths, each containing an *IIviRFSigGen* interface. A class driver object must be extracted from a specific Path interface.

7.5 The IVI Event Server

The *IVI Event Server* (also part of the *IVI Shared Components*) is the central transfer point for events exchanged between IVI drivers and IVI applications.
In the case of the R&S SMU it is used to pass traceing records from the IVI driver to the R&S event logger (RsIviLog.exe). To enable traceing, use

```
[Visual Basic and VB Script]
Obj.System.TraceEnabled = True
```

```
[C++]
Obj->GetSystem()->PutTraceEnabled(VARIANT_TRUE);
```

```
[Ansi C]
ISmuSystem* System;
IRsSmu_get_System(Obj, &System);
ISmuSystem_put_TraceEnabled(System, VARIANT_TRUE);
```

The R&S event logger then displays the *SCPI* commands executed by the driver.