ROHDE&SCHWARZ FS300 Spectrum Analyzer 9kHz...3GHz
R&S Smart instruments
The new product family from Rohde & Schwarz

Remote Control Manual
Series300 Spectrum Analyzer

VXI Plug & Play Style Instrument Driver

ROHDE & SCHWARZ GmbH & Co. KG
Test and Measurement Division
Mühldorfstraße 15
81671 München, Germany

4th edition 03/2006
Instrument Driver Revision 2.4, 03/2006

Table of Contents

About this Manual

Information

This manual is intended to provide you with all the information that is necessary for remote control of the Rohde&Schwarz Series300 Spectrum Analyzer via VXI Plug & Play style Instrument Driver.

Why Use Instrument Drivers?

Information

Many Rohde&Schwarz customers prefer the graphical programming languages LabVIEW from National Instruments or VEE from Agilent when writing applications for T&M equipment. Quite often, C-based LabWindows/CVI from National Instruments and Visual Basic or Visual C++ from Microsoft/ Borland is also used.

As a service, Rohde&Schwarz provides software device drivers free of charge for all these programming languages. All recent T&M equipment is supported, and often demo programs are also available.

Writing an application can take a lot of time. But writing the application is not the end of the story, since the T&M device must also be driven. With complex equipment this is a time-consuming task, since just the description of the register command set may comprise several hundred pages and assumes a detailed knowledge about the instrument’s hardware. This is the reason why Rohde&Schwarz provides ready-to-use software device drivers for all major interfaces, relieving designers of these efforts to a great extent. Extensive search for commands in manuals can be avoided because the work has already been invested when developing the driver.

Manual Concept

About this chapter Chapter 1 contains information for the user/installer of the Rohde&Schwarz Series300 Spectrum Analyzer VXI Plug & Play style Instrument Driver.

More Information Chapter 0 describes the function tree layout of the Series300 Spectrum Analyzer

1.1 Introduction

Introduction The Rohde&Schwarz Series300 Spectrum Analyzer drivers are single 32-bit drivers.

This Rohde&Schwarz Series300 Spectrum Analyzer driver conforms to some parts of the VXI Plug & Play driver standard, which are applicable to conventional GPIB and other non-VXI instruments (that is, rack and stack instruments). The formal VXI Plug & Play standard only covers VXI Instruments and some elements of the standard do not apply to the Rohde&Schwarz Series300 Spectrum Analyzer since it is not a VXI instrument. One of the differences is, that there is no soft front panel, as the Rohde&Schwarz Series300 Spectrum Analyzer can be controlled from its hardware front panel.

Options of the driver:

1. Conformance with the VXI Plug & Play standard. The only exception is that it does not have a soft front panel.
2. It is not built on top of, and does not use the services provided by VISA. VISA is used for its prototype definitions and future compatibility with the other VXIplug&play drivers. If VISA library is not available, then rssitype.h provides data type definitions.
3. It includes a "Function Panel" (.fp) file, which allows it to be used with visual programming environments such as HP-VEE, LabWindows/CVI, and LabVIEW.
4. It includes a comprehensive on-line help file, which complements the instrument manual. The help file presents detailed documentation of each function.
5. The programming sources are included so that the driver can be modified if needed. The source conforms to VXI Plug & Play standards. Modifications should only be carried out by people who are familiar with the VXIplug&play standard.
6. It includes a Visual Basic include file (.bas) which contains the function calls in Visual Basic syntax, so that driver functions can be called from Visual Basic. If you use Visual Basic with this driver, you should be familiar with C/C++ function declarations. In particular, care must be taken when working with C/C++ pointers.

1.2 Instrument-Specific Information

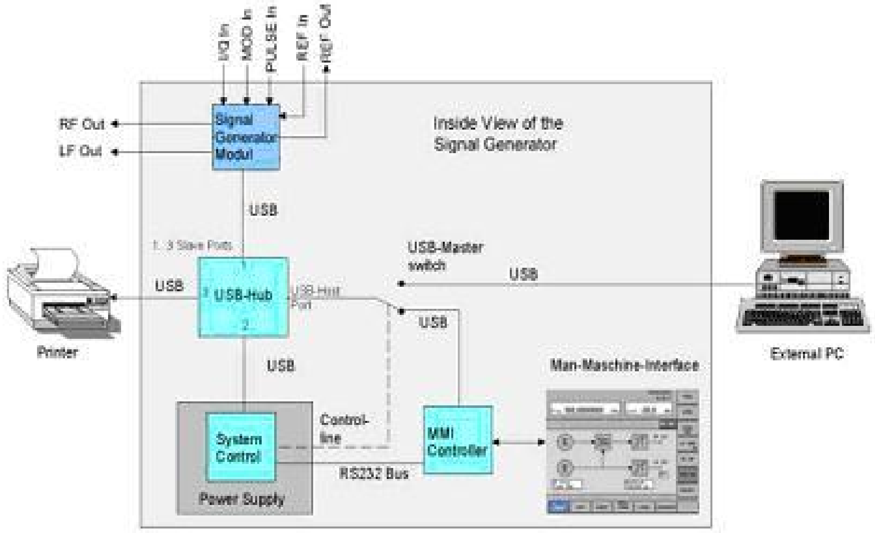
Specific Information

The Series300 Spectrum Analyzer instrument driver is used for remote control of device(s) connected via USB bus. The distribution package (installer) provides the host computer with all the support files necessary to be able to establish a communication session between the host computer and device. The installation process is self-guided.

Instrument Description

The Series300 Spectrum Analyzer consists of two USB instruments, i.e. measurement module, and the system controller associated with the instrument platform in the power supply. The VXI plug&play driver provides the possibility to communicate directly with the Series300 measurement module.

Internal Structure of a Series 300 device



1.2.1 Instrument Addresses (Resource Strings)

Uppercase letters

When using VXI Plug & Play instrument drivers, instrument addresses should be written completely in uppercase letters. Implementation of the addressing scheme is vendor specific and some vendors support mixed cases. However, for maximum portability, the instrument address should use uppercase characters only.

Instrument Descriptor

Based on the Resource Descriptor (instrument address, resource string), the driver establishes a communication session with a device. The syntax of the Resource Descriptor is shown below. Optional parameters are shown in square brackets ([]).

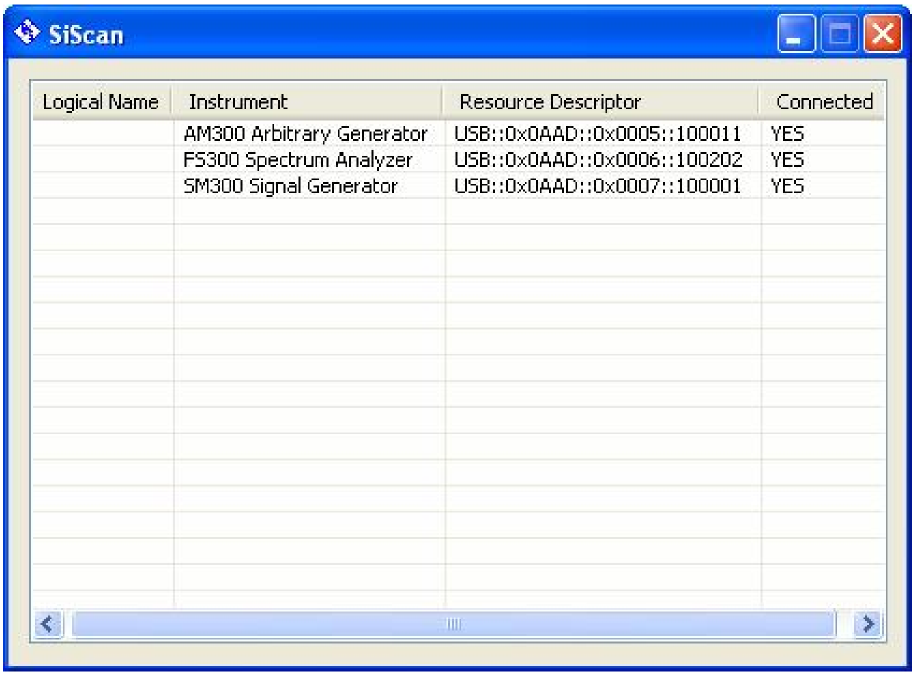
USB::manufacturer_ID::model_code::serial_number

(The **BOLD** printed parameters of the resource string are fixed. Only the serial number will change, if you are using a different FS300).

Brief description of the resource string components:

- **USB** denotes used interface
- **Manufacturer ID** is 0xAAD for Rohde&Schwarz
- **Model Code** is the product identification (0x6 for FS300 Spectrum Analyzer or 0x28 for FS315 Spectrum Analyzer)
- **Serial Number** is the serial number printed on the instrument box

Resource descriptors of the connected devices can be taken using build in utility SiScan.



Logical Name	Instrument	Resource Descriptor	Connected
	AM300 Arbitrary Generator	USB::0x0AAD::0x0005::100011	YES
	FS300 Spectrum Analyzer	USB::0x0AAD::0x0006::100202	YES
	SM300 Signal Generator	USB::0x0AAD::0x0007::100001	YES

1.2.2 Using Callbacks

⚠ **Caution**	Callbacks are not supported with this driver.

1.2.3 Thread Safety

⚠ **Caution**	We recommend not using multiple threads to communicate with the instrument in parallel.

1.2.4 Device Identification and Logical Names

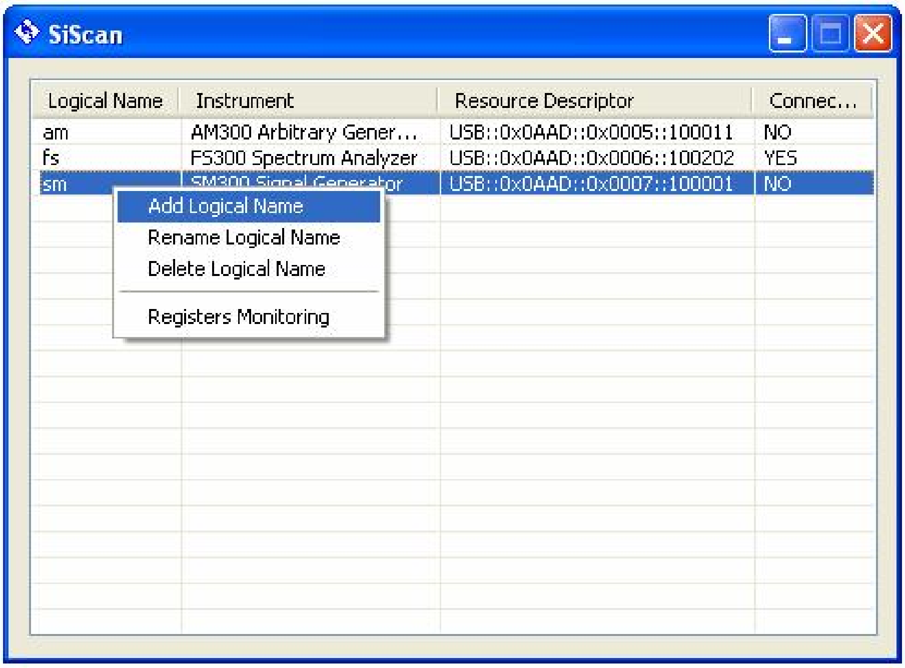
SiScan

For easy identification of devices on the USB bus use the ***SiScan*** application. ***SiScan*** is distributed with the driver and stored in the system's program folder (typically C:\Program Files\Series300\SiTools).

The instrument driver supports logical names as aliases of resource strings. You can pass logical name instead of instrument descriptor (resource string).

For example: device_1 instead of USB::0x0aad::0x6::123456.

Logical names can be configured with the ***SiScan*** application.



Logical Name	Instrument	Resource Descriptor	Connec...
am	AM300 Arbitrary Gener...	USB::0x0AAD::0x0005::100011	NO
fs	FS300 Spectrum Analyzer	USB::0x0AAD::0x0006::100202	YES
sm	SM300 Signal Generator	USB::0x0AAD::0x0007::100001	NO

Windows registry

All the logical names are accessible under the Windows system registry key:

HKEY_LOCAL_MACHINE\SOFTWARE\Rohde&Schwarz\SiControl

Logical name record comprises the following items:

- **BoxResource**, which is the resource descriptor used with device specific drivers (VXIpnp style instrument drivers)
- **ModuleResource** is the resource descriptor used to open session with specified module with its serial number (low-level function SiOpenDevice)
- **ModelName** is the string descriptor of the device used to make the registry entry more readable for the user

Logical names are passed as alphanumeric strings. It is allowed to use more than one logical name for one device.

1.2.5 Hot Plug & Unplug Support

Description

Device can be set to local mode (unplugged) and then back to remote mode (plugged) without loosing the initialized communication session. Once the session is opened (rssifs_init), session based data are safe until the session is closed (rssifs_close).

Communication session

A session is a communication path between a software element on host computer and a resource (instrument). Every communication session is unique. It is allowed to open up to 256 sessions per device.

1.2.6 SiTools

Description

A set of the utilities called ***SiTools*** is used to manage (***SiScan***) or monitor (***SiMonitor***) connected devices. All the respective utilities are stored under a path defined in the windows registry under

HKEY_LOCAL_MACHINE\SOFTWARE\Rohde&Schwarz\SiTools

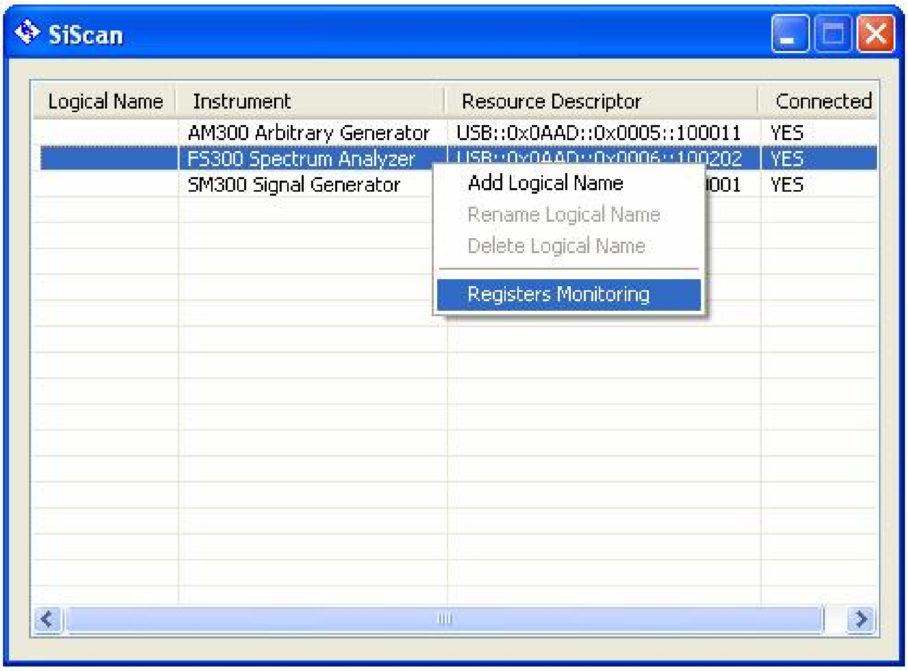
key (***SiToolsDir***). Each of ***SiTools*** provides comprehensive help how to use it. ***SiTools*** are installed with the driver and stored in the system's program folder (typically C:\Program Files\Series300\SiTools).

Note

Please note that the ***SiTools*** are not necessary for the system.

SiScan

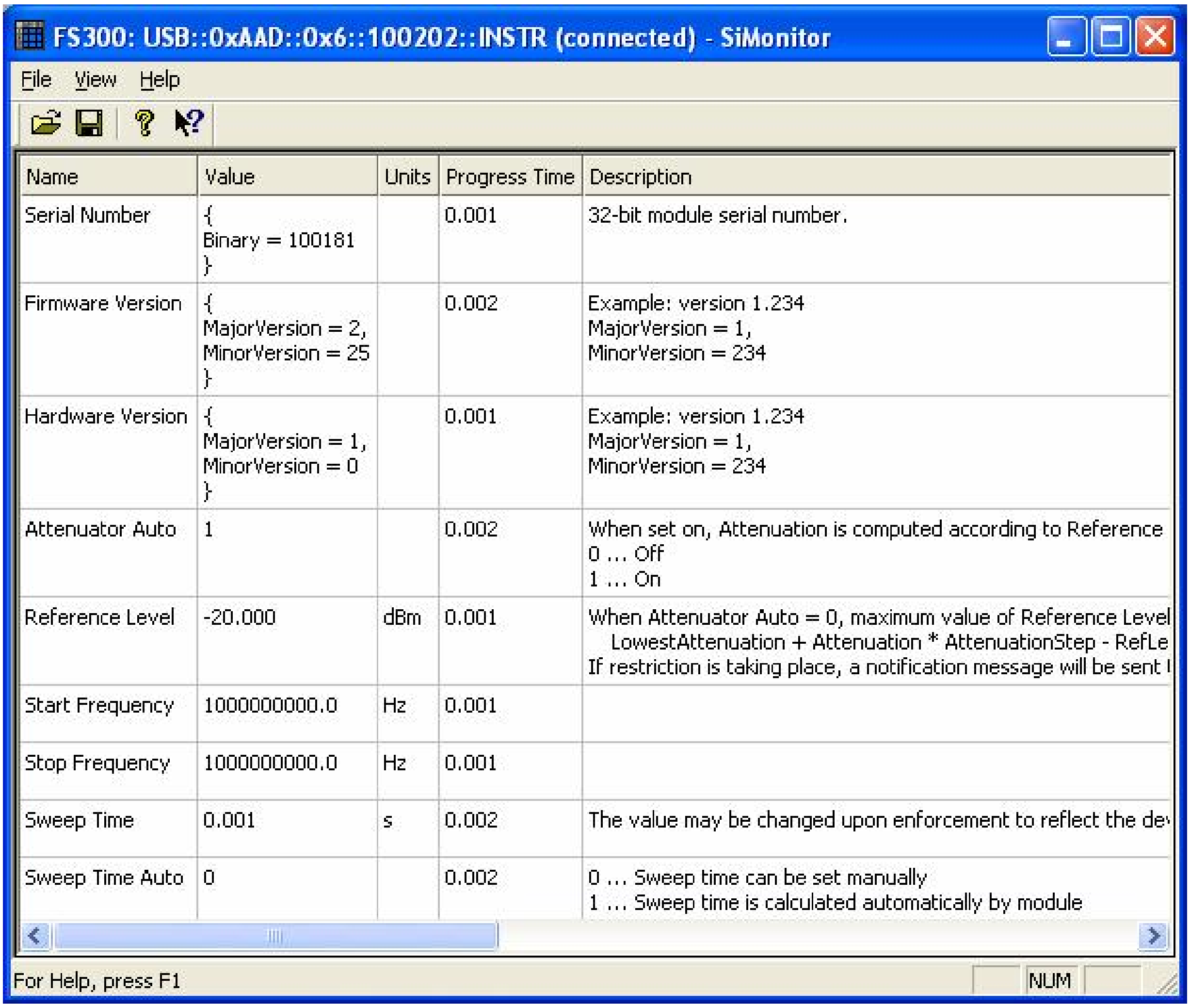
SiScan is a tool providing also access to the low-level monitoring tool called ***SiMonitor***. ***SiMonitor*** can be launched from ***SiScan*** by mouse right-click on the connected device in the list.



Logical Name	Instrument	Resource Descriptor	Connected
	AM300 Arbitrary Generator	USB::0x0AAD::0x0005::100011	YES
	FS300 Spectrum Analyzer	USB::0x0AAD::0x0006::100202	YES
	SM300 Signal Generator	001	YES

SiMonitor

The ***SiMonitor*** is used to provide information of current device settings. All parameters accessible in the table are core configuration elements of monitored device. Since polling these parameters affects the performance of other applications communicating with the instrument, it would be better to use it only for debugging and maintenance during the development period of the remote control application. More detailed information on using the ***SiMonitor*** can be found in the associated help file.



FS300: USB::0xAAD::0x6::100202::INSTR (connected) - SiMonitor

Name	Value	Units	Progress Time	Description

Serial Number	{
	Binary = 100181
	}
			0.001	32-bit module serial number.

Firmware Version	{
	MajorVersion = 2,
	MinorVersion = 25
	}
			0.002	Example: version 1.234
				MajorVersion = 1,
				MinorVersion = 234

Hardware Version	{
	MajorVersion = 1,
	MinorVersion = 0
	}
			0.001	Example: version 1.234
				MajorVersion = 1,
				MinorVersion = 234

Attenuator Auto	1		0.002	When set on, Attenuation is computed according to Reference 0 ... Off 1 ... On
Reference Level	-20.000	dBm	0.001	When Attenuator Auto = 0, maximum value of Reference Level LowestAttenuation + Attenuation * AttenuationStep - RefLe If restriction is taking place, a notification message will be sent
Start Frequency	1000000000.0	Hz	0.001	
Stop Frequency	1000000000.0	Hz	0.001	
Sweep Time	0.001	s	0.002	The value may be changed upon enforcement to reflect the de
Sweep Time Auto	0		0.002	0 ... Sweep time can be set manually 1 ... Sweep time is calculated automatically by module

For Help, press F1

1.3 Using An Instrument Driver in Application Development Environments

This section offers suggestions on using the rssifs_32.dll within various application development environments.

Note

The application notes “R&S SmartInstruments^{TM} Family 300 Basic Programming Guide”, which is available on the Rohde&Schwarz homepage, will also give a detailed overview about who to use the drivers in different development environments.

1.3.1 Microsoft Visual C++ 4.0 (or higher) and Borland C++ 4.5 (or higher)

Refer to your Microsoft Visual C++ or Borland C++ manuals for information on linking and calling DLLs.

1. The driver uses Pascal calling conventions.
2. Rebuilding the driver DLL should be done in a different directory than the one the driver was installed in order to differentiate the changes.

1.3.2 Microsoft Visual Basic 5.0 (or higher)

Refer to the Microsoft Visual BASIC manual for information on calling DLLs. The BASIC include file is rssifs.bas, which is contained in the directory ~vxipnp\winnt\include. The ~ refers to the directory in the VXIPNP variable. By default this is set to C:\. You may also need to include the visa.bas file that comes with your VISA DLL.

1.3.3 HP VEE Version 3.2 (or higher)

Your copy of HP VEE for WINDOWS contains a document titled "Using VXI plug&play Drivers with HP VEE for Windows". This document contains the detailed information you need for HP VEE applications.

1.3.4 National Instruments LabWindows/CVI(R) 4.0.1 (or higher)

The FS300 Spectrum Analyzer driver is supplied as both a source code file and as a dynamic link library file (dll). There are several advantages to using the dll form of the driver. These include:

1. Transportability across different computer platforms
2. Faster load time for your project

LabWindows/CVI (R) by default will attempt to load the source version of the instrument driver. To load the dll you must include the file rssifs.fp in your project. This file can be found in the vxipnp\winnt\rssifs directory. Do not include rssifs.c in your project. You must also provide an include path for rssifs.h. This is done by adding the directory ~vxipnp\winnt\include to the include paths (CVI Project Option menu) if you have not already done so. The ~ refers to the directory in the VXIPNP variable. By default this is set to C:\.

1.3.5 National Instruments LabVIEW(R) 6.1 (or higher)

If you want to use this driver as a standard LabVIEW driver, please copy the content of ~VXIpnp\GWinnt\rssifs directory into your LabVIEW directory (~LabVIEW\instr.lib\rssifs\) manually. The driver will then be directly accessible from the LabVIEW Instrument Driver function palette menu.

1.4 VXIPNP Directory Location

The driver does not use VISA library, but it is installed to the VXIpnp directory and can be used as a standard VXIpnp instrument driver. If VISA library is not installed on the target machine, then installer will create a directory structure to fit with VXIpnp standard.

1.5 Files Installed

The install program will place the following files on the hard drive:

rssifs.h	Header file for use with C, HP VEE and LabView/LabWindows
rssi_fs300.h	FS300 specific header file
rssi_fsxxx.h	FS315 specific header file
rssifs.c	Source code for use with C
rssi_fs300.c	FS300 specific source code file
rssi_fsxxx.c	FS315 specific source code file
rssifs.def	Definition file for use with C++ when building the .dll file
rssifs.fp	Function Panel file for use with HP VEE and LabVIEW/LabWindows
rssifs.bas	Module file for use with Visual BASIC
rssifs.vb	Module file for use with .NET BASIC
rssifs.hlp	Help file for use with VB
rssifs.chm	Compressed HTML help for use with C and Visual Basic
rssifs.lib	Library file for use with C++
rssifs_32.dll	Dynamic Link Library file for use with all platforms
instrsup.dll	Instrument support Dynamic Link Library file from LabWindows/CVI.
SiControl.dll	Dynamic Link Library file for use with all platforms
SiControl.lib	Library file for use with C/C++
SiControl.h	Header file for use with C and LabWindows
rssitype.h	Header file for use with C and LabWindows (VISA data types)
rssi.inf	RSSI (USB I/O) Setup Information file
rssi.sys	RSSI (USB I/O) Driver file
readme.txt	File that contains general information
license.pdf	Instrument Driver License Agreement
SiTools	Set of device management utilities (SiMonitor, SiScan)

Table 0-1: Program Installation

LabVIEW installer in addition will place the following files on the hard drive:

rssifs.llb	LabVIEW library containing the driver VIs
rssifs.chm	LabVIEW Context Help (LabVIEW 6.1 or higher)
*.mnu	LabVIEW palette menu files of the driver

Table 0-2: LabVIEW Installation

Note If a particular platform is not going to be used, the corresponding platform-specific files may be deleted. Installer may bring more files than listed in above section.

Programmer's Reference Manual

1.6 Instrument Driver Tree Structure

Class/Panel Name	Function Name
Initialization	rssifs_init
Application Functions	
Read Spectrum	rssifs_appReadSpectrum
Configuration Functions	
Frequency Settings	
Configure Start Stop Frequency	rssifs_confStartStopFrq
Configure Span Center Frequency	rssifs_confSpanCenterFrq
Configure Frequency Offset	rssifs_confFreqOffset
Configure Signal Track	rssifs_confSignalTrack
Low-Level Functions	
Get Center Frequency	rssifs_getCenterFrequency
Get Frequency Span	rssifs_getFrequencySpan
Get Frequency Offset	rssifs_getFrequencyOffset
Get Start Frequency	rssifs_getStartFrequency
Get Stop Frequency	rssifs_getStopFrequency
Get Signal Track	rssifs_getSignalTrack

Class/Panel Name	Function Name

Get Signal Track

C Function Prototype

```
ViStatus rssifs_getSignalTrack (
      ViSession instrumentHandle,
      ViInt32* signalTrack,
      ViReal64* bandwidthValue,
      ViReal64* thresholdValue);
```

Basic Function Prototype

```
Function rssifs_getSignalTrack (
      ByVal instrumentHandle As ViSession,
      signalTrack As ViInt32,
      bandwidthValue As ViReal64,
      thresholdValue As ViReal64) As ViStatus
```

Purpose

This function returns parameters of the signal-track (the track bandwidth, threshold and state).

Parameters List

1. **ViSession instrumentHandle [in]**

   This control accepts the Instrument Handle returned by the Initialize function to select the desired instrument driver session.

   Default Value: None

2. **ViInt32 signalTrack [out]**

   Returns whether signal track operation is active or not.

   Valid Range:

   - 0 - Signal Track Off
   - 1 - Signal Track On

3. **ViReal64 bandwidthValue [out]**

   Returns bandwidth around the center frequency within which the largest signal is searched (in Hz).

4. **ViReal64 thresholdValue [out]**

   Returns the threshold above which the largest signal is searched for (in dBm).

Return Value

Returns the status code of this operation.

The meaning of the status code is described in section Error (Status) Codes.

Amplitude Settings

Class/Panel Name	Function Name
Configure Reference Level	rssifs_confRefLevel
Configure Reference Level Offset	rssifs_confRefLevelOffset
Low-Level Functions	
Get Reference Level	rssifs_getReferenceLevel
Get Reference Level Offset	rssifs_getReferenceLevelOffset
Input Settings	
Configure RF Input Attenuation	rssifs_confRFInAtt
Configure RF Input Attenuator Auto	rssifs_confRFInAttAuto

Class/Panel Name	Function Name
Configure RF Input High Sensitivity	rssifs_confRFInHighSensitivity
Low-Level Functions	
Get RF Input Attenuation	rssifs_getRFInputAttenuation
Get RF Input Attenuator Mode	rssifs_getRFInputAttenuatorMode
Marker Settings	
Configure Marker State	rssifs_confMarkState
Configure Delta Marker State	rssifs_confDeltaMarkState

Configure Delta Marker State

C Function Prototype

```
ViStatus rssifs_confDeltaMarkState (
            ViSession instrumentHandle,
            ViInt32 deltaMarkerNumber,
            ViBoolean deltaMarkerState);
```

Basic Function Prototype

```
Function rssifs_confDeltaMarkState (
       ByVal instrumentHandle As ViSession,
       ByVal deltaMarkerNumber As ViInt32,
       ByVal deltaMarkerState As ViBoolean) As ViStatus
```

Purpose

This function is used to enable or disable selected delta marker for its operation over the trace cache data.

Parameters List

1. **ViSession instrumentHandle [in]**

   This control accepts the Instrument Handle returned by the Initialize function to select the desired instrument driver session.

   Default Value: None

2. **ViInt32 deltaMarkerNumber [in]**

   Selects delta marker to configure.

   Valid Value: 1 to 2

   Default Value: 1

3. **ViBoolean deltaMarkerState [in]**

   Enables or disables selected delta marker.

   Valid Range:
   VI_FALSE (0) - Off (Default Value)
   VI_TRUE (1) – On

Return Value

Returns the status code of this operation.

The meaning of the status code is described in section Error (Status) Codes.

Configure Marker Position	rssifs_confMarkPosition
Configure Delta Marker Position	rssifs_confDeltaMarkPosition

Class/Panel Name	Function Name
Configure Delta Marker Position	

C Function Prototype

```
ViStatus rssifs_confDeltaMarkPosition (
      ViSession instrumentHandle,
      ViInt32 deltaMarkerNumber,
      ViReal64 deltaMarkerPosition);
```

Basic Function Prototype

```
Function rssifs_confDeltaMarkPosition (
      ByVal instrumentHandle As ViSession,
      ByVal deltaMarkerNumber As ViInt32,
      ByVal deltaMarkerPosition As ViReal64) As ViStatus
```

Purpose

This function positions selected delta marker to the indicated frequency (span > 0) or time (span = 0) relative to the corresponding normal marker position.

Parameters List

1. **ViSession instrumentHandle [in]**

   This control accepts the Instrument Handle returned by the Initialize function to select the desired instrument driver session.

   Default Value: None

2. **ViInt32 deltaMarkerNumber [in]**

   Selects delta marker to configure.

   Valid Value: 1 to 2

   Default Value: 1

3. **ViReal64 deltaMarkerPosition [in]**

   Sets selected delta marker to the specified position relative to the corresponding normal marker position.

   Valid Range:

   - Frequency (span > 0) (offset = 0.0):
     FS300/FS315: 0.0 Hz to 3.0e9 Hz
   - Time (span = 0):
     FS300: 0.0 s to 20.0 s
     FS315: 0.0 s to 10.0 s

   Default Value: 0.0

Note

Frequency offset value applies only when (span > 0).

Return Value

Returns the status code of this operation.

The meaning of the status code is described in section Error (Status) Codes.

Configure Marker Frequency Counter	rssifs_confMarkFreqCnt
Configure Marker Peak Excursion	rssifs_confMarkPeakExcursion
Configure Marker Search Mode	rssifs_confMarkSearchMode
Marker Search	rssifs_actMarkSearch

Class/Panel Name	Function Name
Marker Search N dB Down	rssifs_confMarkerSearchNdBDown

Marker Search N dB Down

C Function Prototype

```
ViStatus rssifs_confMarkerSearchNdBDown (
   ViSession instrumentHandle,
   ViInt32 markerNumber,
   ViReal64 ndBDownValue,
   ViBoolean ndBDown);
```

Basic Function Prototype

```
Function rssifs_confMarkerSearchNdBDown (
   ByVal instrumentHandle As ViSession,
   ByVal markerNumber As ViInt32,
   ByVal ndBDownValue As ViReal64,
   ByVal ndBDown As ViBoolean) As ViStatus
```

Purpose

This function configures the measurement of the temporary markers which are n dB below the active reference marker.

Parameters List

1. **ViSession instrumentHandle [in]**

   This control accepts the Instrument Handle returned by the Initialize functionto select the desired instrument driver session.

   Default Value: None

2. **ViInt32 markerNumber [in]**

   Selects the marker to configure.

   Valid Value: 1 to 2

   Default Value: 1

3. **ViReal64 ndBDownValue [in]**

   Enters the N dB down value.

   Valid Range: 0.0 dB to 100.0 dB

   Default Value: 3.0 dB

4. **ViBoolean ndBDown [in]**

   Activates and deactivates temporary markers which are located n dB below the active reference marker.

   Valid Range:
   - VI_FALSE (0) - Off (Default Value)
   - VI_TRUE (1) - On

Return Value

Returns the status code of this operation.

The meaning of the status code is described in section Error (Status) Codes.

Low-Level Functions

Get Marker State	rssifs_getMarkerState
Get Delta Marker State	rssifs_getDeltaMarkState

Class/Panel Name	Function Name
Get Delta Marker State	

C Function Prototype

```
ViStatus rssifs_getDeltaMarkerState (
      ViSession instrumentHandle,
      ViInt32 deltaMarkerNumber,
      ViBoolean* deltaMarkerState);
```

Basic Function Prototype

```
Function rssifs_getDeltaMarkerState (
      ByVal instrumentHandle As ViSession,
      ByVal deltaMarkerNumber As ViInt32,
      deltaMarkerState As ViBoolean) As ViStatus
```

Purpose

This function returns the state of selected delta marker.

Parameters List

1. **ViSession instrumentHandle [in]**

   This control accepts the Instrument Handle returned by the Initialize function to select the desired instrument driver session.

   Default Value: None

2. **ViInt32 deltaMarkerNumber [in]**

   Selects delta marker to configure.

   Valid Value: 1 to 2

   Default Value: 1

3. **ViBoolean deltaMarkerState [out]**

   Returns the state of selected delta marker.

   Valid Values:
   - VI_FALSE (0) – Off
   - VI_TRUE (1) - On

Return Value

Returns the status code of this operation.

The meaning of the status code is described in section Error (Status) Codes.

Class/Panel Name	Function Name
Get Marker Position	rssifs_getMarkerPosition
Get Delta Marker Position	rssifs_getDeltaMarkPosition

Class/Panel Name	Function Name
Get Delta Marker Position	

C Function Prototype

```
ViStatus rssifs_getDeltaMarkPosition (
      ViSession instrumentHandle,
      ViInt32 deltaMarkerNumber,
      ViReal64* deltaMarkerPosition);
```

Basic Function Prototype

```
Function rssifs_getDeltaMarkPosition (
      ByVal instrumentHandle As ViSession,
      ByVal deltaMarkerNumber As ViInt32,
      deltaMarkerPosition As ViReal64) As ViStatus
```

Purpose

This function returns positions of selected delta marker relative to the corresponding normal marker position.

Parameters List

1. **ViSession instrumentHandle [in]**

   This control accepts the Instrument Handle returned by the Initialize function to select the desired instrument driver session.

   Default Value: None

2. **ViInt32 deltaMarkerNumber [in]**

   Selects delta marker to configure.

   Valid Value: 1 to 2

   Default Value: 1

3. **ViReal64 deltaMarkerPosition [out]**

   Returns position of selected delta marker relative to the corresponding normal marker position. Value is represented as frequency (span > 0) or time (span = 0).

Return Value

Returns the status code of this operation.

The meaning of the status code is described in section Error (Status) Codes.

Get Marker Frequency Counter State	rssifs_getMarkerFreqCounterState
Get Marker Frequency Counter Resolution	rssifs_getMarkerFreqCounterResolution
Get Marker Peak Excursion	rssifs_getMarkerPeakExcursion
Get Marker Search Mode	rssifs_getMarkSearchMode
Marker Search N dB **Down**	rssifs_getMarkerSearchNdBDown

Class/Panel Name	Function Name

Get Marker Search N dB Down

C Function Prototype

```
ViStatus rssifs_getMarkerSearchNdBDown (
      ViSession instrumentHandle,
      ViInt32 markerNumber,
      ViReal64* ndBDownValue,
      ViBoolean* ndBDown);
```

Basic Function Prototype

```
Function rssifs_getMarkerSearchNdBDown (
      ByVal instrumentHandle As ViSession,
      ByVal markerNumber As ViInt32,
      ndBDownValue As ViReal64,
      ndBDown As ViBoolean) As ViStatus
```

Purpose

This function returns measurement settings of the temporary markers which are n dB below the active reference marker.

Parameters List

1. **ViSession instrumentHandle [in]**

   This control accepts the Instrument Handle returned by the Initialize function to select the desired instrument driver session.

   Default Value: None

2. **ViInt32 markerNumber [in]**

   Selects the marker to configure.

   Valid Value: 1 to 2

   Default Value: 1

3. **ViReal64 ndBDownValue [out]**

   Returns the N dB down value.

4. **ViBoolean ndBDown [out]**

   Returns whether N dB Down marker measurement is enabled or disabled.

   Valid Range:
   - VI_FALSE (0) – Off
   - VI_TRUE (1) – On

Return Value

Returns the status code of this operation.

The meaning of the status code is described in section Error (Status) Codes.

Trace Settings

Configure Trace Mode	rssifs_confTraceMode
Configure Trace Detector	rssifs_confTraceDetector
Configure Trace Unit	rssifs_confTraceUnit

Class/Panel Name	Function Name

Configure Trace **Detector**

C Function Prototype

```
ViStatus rssifs_confTraceDetector (
      ViSession instrumentHandle,
      ViInt32 traceDetector);
```

Basic Function Prototype

```
Function rssifs_confTraceDetector (
      ByVal instrumentHandle As ViSession,
      ByVal traceDetector As ViInt32) As ViStatus
```

Purpose

This function controls the recording of trace values via the type of detector.

Note

This function is available for FS315 only.

Parameters List

1. **ViSession instrumentHandle [in]**

   This control accepts the Instrument Handle returned by the Initialize function to select the desired instrument driver session.

   Default Value: None

2. **ViInt32 traceDetector [in]**

   Defines the trace detector used.

   Valid Range:
   - RSSIFS_TRACE_DETECTOR_MIN_PEAK (1) - Min Peak
   - RSSIFS_TRACE_DETECTOR_MAX_PEAK (2) - Max Peak
   - RSSIFS_TRACE_DETECTOR_SAMPLE (3) – Sample
   - RSSIFS_TRACE_DETECTOR_RMS (4) – RMS
   - RSSIFS_TRACE_DETECTOR_AVG (5) - Average

   Default Value: RSSIFS_TRACE_DETECTOR_MAX_PEAK (2)

Note

- Min Peak:
  The min peak detector selects from the samples allocated to a pixel the one with the minimum value.

- Max Peak:
  The max peak detector selects from the samples allocated to a pixel the one with the maximum value.

- Sample:
  The sample detector samples the IF envelope for each pixel of the trace to be displayed only once.

- RMS:
  The RMS (root mean square) detector calculates the power for each pixel of the displayed trace from the samples allocated to a pixel. The result corresponds to the signal power within the span represented by the pixel.

- Average:
  The average detector calculates the linear average for each pixel of the displayed trace from the samples allocated to a pixel.

Return Value

Returns the status code of this operation.

The meaning of the status code is described in section Error (Status) Codes.

1.6.1.1.1 Configure Trace Unit

C Function Prototype

```
ViStatus rssifs_confTraceUnit (
      ViSession instrumentHandle,
      ViInt32 traceUnit);
```

Class/Panel Name	Function Name
Get Trace Mode	rssifs_getTraceMode
Get Trace Detector	rssifs_getTraceDetector
Get Trace Unit	rssifs_getTraceUnit
Demodulator Settings	
Configure Demodulator	rssifs_confDemodulator
Configure Demodulator Volume	rssifs_confDemodulatorVolume
Configure Demodulator Appearance	rssifs_confDemodulatorAppearance
Low-Level Functions	
Get Demodulator State	rssifs_getDemodulatorState
Get Demodulator Type	rssifs_getDemodulatorType
Get Demodulator Volume	rssifs_getDemodulatorVolume
Get Demodulator Time	rssifs_getDemodulatorTime
Get Demodulator Display	rssifs_getDemodulatorDisplay
Tracking Generator Settings	
Configure Tracking Generator	rssifs_confTrackingGenerator
Configure Tracking Generator Level	rssifs_confTrackingGeneratorLevel
Configure Tracking Generator Frequency	rssifs_confTrackingGeneratorFrequency
Low-Level Functions	
Get Tracking Generator State	rssifs_getTrackingGeneratorState
Get Tracking Generator Level	rssifs_getTrackingGeneratorLevel
Get Tracking Generator Frequency	rssifs_getTrackingGeneratorFrequency

Class/Panel Name	Function Name

Get Trace Detector

C Function Prototype

```
ViStatus rssifs_getTraceDetector (
      ViSession instrumentHandle,
      ViInt32* traceDetector);
```

Basic Function Prototype

```
Function rssifs_getTraceDetector (
      ByVal instrumentHandle As ViSession,
      traceDetector As ViInt32) As ViStatus
```

Purpose

This function returns the trace detector used.

Note

This function is available for FS315 only.

Parameters List

1. **ViSession instrumentHandle [in]**

   This control accepts the Instrument Handle returned by the Initialize function to select the desired instrument driver session.

   Default Value: None

2. **ViInt32 traceDetector [out]**

   Returns the trace detector used.

   Valid Range:

   - RSSIFS_TRACE_DETECTOR_MIN_PEAK (1) - Min Peak
   - RSSIFS_TRACE_DETECTOR_MAX_PEAK (2) - Max Peak
   - RSSIFS_TRACE_DETECTOR_SAMPLE (3) – Sample
   - RSSIFS_TRACE_DETECTOR_RMS (4) – RMS
   - RSSIFS_TRACE_DETECTOR_AVG (5) - Average

Return Value

Returns the status code of this operation.

The meaning of the status code is described in section Error (Status) Codes.

1.6.1.1.1.1 Get Trace Unit

C Function Prototype

```
ViStatus rssifs_getTraceUnit (
      ViSession instrumentHandle,
      ViInt32* traceUnit);
```

Basic Function Prototype

```
Function rssifs_getTraceUnit (
      ByVal instrumentHandle As ViSession,
      traceUnit As ViInt32) As ViStatus
```

Purpose

This function returns current trace data unit.

Parameters List

1. **ViSession instrumentHandle [in]**

   This control accepts the Instrument Handle returned by the Initialize function to select the desired instrument driver session.

   Default Value: None

2. **ViInt32 traceUnit [out]**

   Returns current trace data unit.

   Valid Range:

   - RSSIFS_TRACE_UNIT_VOLT (0) - Volt
   - RSSIFS_TRACE_UNIT_WATT (1) - Watt
   - RSSIFS_TRACE_UNIT_DBM (2) - dBm
   - RSSIFS_TRACE_UNIT_DBMV (3) - dBmV
   - RSSIFS_TRACE_UNIT_DBUV (4) - dBuV
   - RSSIFS_TRACE_UNIT_DBUA (5) - dBuA

Return Value

Returns the status code of this operation.

Class/Panel Name	Function Name
Configure Sweep	rssifs_confSweep
Configure Sweep Time	rssifs_confSweepTime
Configure Sweep Points	rssifs_confSweepPoints
Low-Level Functions	
Get Sweep Count	rssifs_getSweepCount
Get Sweep Time	rssifs_getSweepTime
Get Sweep Mode	rssifs_getSweepMode
Get Sweep Points	rssifs_getSweepPoints
Trigger Settings	
Configure Trigger	rssifs_confTrg
Configure Trigger Delay	rssifs_confTrgDelay
Low-Level Functions	
Get Trigger Delay	rssifs_getTriggerDelay
Get Trigger Source	rssifs_getTriggerSource
Get Trigger Level	rssifs_getTriggerLevel
Get Trigger Slope	rssifs_getTriggerSlope
Bandwidth Settings	
Configure Bandwidth	rssifs_configureBandwidth
Configure Resolution Bandwidth	rssifs_confResBW
Configure Video Bandwidth	rssifs_confVideoBW
Configure RBW vs Span Coupling	rssifs_configureRBWSpanCoupling
Configure RBW vs VBW Coupling	rssifs_configureRBWVBWCoupling
Low-Level Functions	
Get Resolution Bandwidth	rssifs_getResolutionBandwidth
Get Video Bandwidth	rssifs_getVideoBandwidth
Get RBW vs Span Coupling Mode	rssifs_getRBWSpanCouplingMode
Get RBW vs VBW Coupling	rssifs_getRBWVBWCoupling

Class/Panel Name	Function Name
Get RBW vs VBW Coupling	

C Function Prototype

```
ViStatus rssifs_getRBWVBWCoupling (
      ViSession instrumentHandle,
      ViReal64* couplingRatio);
```

Basic Function Prototype

```
Function rssifs_getRBWVBWCoupling (
      ByVal instrumentHandle As ViSession,
      couplingRatio As ViReal64) As ViStatus
```

Purpose

This function returns the automatic coupling between the resolution bandwidth (RBW) and video bandwidth (VBW).

Parameters List

1. **ViSession instrumentHandle [in]**
   This control accepts the Instrument Handle returned by the Initialize function to select the desired instrument driver session.

   Default Value: None

2. **ViReal64 couplingRatio [out]**
   Returns coupling ratio between resolution bandwidth (RBW) and video bandwidth (VBW) (RBW/VBW).

Return Value

Returns the status code of this operation.

The meaning of the status code is described in section Error (Status) Codes.

System Settings	
Configure Reference Oscillator Source	rssifs_confReferenceOsc
Configure Transducer Factor	rssifs_confTransducerFactor
Configure Transducer Factor Values	rssifs_confTransducerFactorValues
Low-Level Functions	
Get Transducer Factor	rssifs_getTransducerFactor
Get Transducer Factor Values	rssifs_getTransducerFactorValues
Measurement Functions	
Configure Channel Power Measurement	rssifs_confChannelPowerMeasurement
Configure Occupied Bandwidth Measurement	rssifs_confOccupiedBandwidthMeasurement
Configure Time Domain Power Measurement	rssifs_confTimeDomainPowerMeasurement
Configure Limit Lines	rssifs_confLimitLines
Low-Level Functions	
Get Channel Power Measurement	rssifs_getChannelPowerMeasurement

Class/Panel Name	Function Name
Get Occupied Bandwidth Measurement	rssifs_getOccupiedBandwidthMeasurement
Get Time Domain Power Measurement	rssifs_getTimeDomainPowerMeasurement
Get Limit Lines	rssifs_getLimitLines

Class/Panel Name	Function Name

Configure Transducer Factor

C Function Prototype

```
ViStatus rssifs_confTransducerFactor (
      ViSession instrumentHandle,
      ViBoolean transducerFactors);
```

Basic Function Prototype

```
Function rssifs_confTransducerFactor (
      ByVal instrumentHandle As ViSession,
      ByVal transducerFactors As ViBoolean) As ViStatus
```

Purpose

This function activates or deactivated usage of transducer factors.

Parameters List

1. **ViSession instrumentHandle [in]**
   This control accepts the Instrument Handle returned by the Initialize function to select the desired instrument driver session.

   Default Value: None

2. **ViBoolean transducerFactors [in]**
   Activates or deactivated usage of transducer factors.

   Valid Range:
   - VI_FALSE (0) - Off (Default Value)
   - VI_TRUE (1) - On

Return Value

Returns the status code of this operation.

The meaning of the status code is described in section Error (Status) Codes.

1.6.1.3.5 Configure Transducer Factor Values

C Function Prototype

```
ViStatus rssifs_confTransducerFactorValues (
      ViSession instrumentHandle,
      ViInt32 noofValues,
      ViReal64[] frequencyValues,
      ViReal64[] levelValues,
      ViInt32 unit);
```

Basic Function Prototype

```
Function rssifs_confTransducerFactorValues (
      ByVal instrumentHandle As ViSession,
      ByVal noofValues As ViInt32,
      frequencyValues As ViReal64,
      levelValues As ViReal64,
      ByVal unit As ViInt32) As ViStatus
```

Purpose

This function is used to define transducer factor values.

Note

- Transducer factors for a sweep are calculated once in advance for every point displayed and are added to the result of the level measurement during the sweep. If the sweep range changes, the correction values are calculated again.
- If the transducer factor is not defined for the entire sweep range, the values missing are replaced by zeroes.
- This function applies over the trace cache data.

Parameters List

1. **ViSession instrumentHandle [in]**
   This control accepts the Instrument Handle returned by the Initialize function to select the desired instrument driver session.

   Default Value: None

Class/Panel Name	Function Name
Trigger Group	
Send Trigger	rssifs_actSendTrg
Send Trigger and Wait for OPC	rssifs_actSendTrgWopc
Abort	rssifs_actAbort
Calibration Group	
Calibration	rssifs_actCalibration
Device Status Group	
Get Device State	rssifs_getDeviceState

Class/Panel Name	Function Name

Device **Status Group**

Description The status reporting system functions provides information on the present operating state of the instrument, e.g. that the instrument is presently ready for immediate operation, working, performing self-test, etc.

1.6.1.4.6 Get Device State

C Function Prototype

```
ViStatus rssifs_getDeviceState (
      ViSession instrumentHandle,
      ViInt32* deviceState);
```

Basic Function Prototype

```
Function rssifs_getDeviceState (
      ByVal instrumentHandle As ViSession,
      deviceState As ViInt32) As ViStatus
```

Purpose This function presents logical state (present operating state) of the device, e.g. that the instrument is presently ready for immediate operation, performing measurement task, performing self-test, etc.

Parameters List

1. **ViSession instrumentHandle [in]**
   This control accepts the Instrument Handle returned by the Initialize function to select the desired instrument driver session.

   Default Value: None

2. **ViInt32 deviceState [out]**
   Returns logical state (present operating state) of the device.

FS300 Spectrum Analyzer		
Value	*Name*	*Description*
0x0	Idle	Device is in power-saving mode and ready for immediate operation.
0x1	Busy	Device is working.
0x80	Sleep	Device is in battery saving mode. It may take some time to wake it up.
0x81	Init	Device is entering Idle mode (waking up from Sleep, booting or performing self test).

FS315 Spectrum Analyzer		
Value	*Name*	*Description*
0x0	Idle	Device is in idle mode and ready for immediate operation.
0x10	Meas	Device performs measurement task.
0x64	Service	Device was triggered to Service mode by service command.
0xFF	Init	Device performs initialization. It is entering Idle mode.
0xFFFF	Sleep	Device is in power-saving mode.

Return Value Returns the status code of this operation.

The meaning of the status code is described in section Error (Status) Codes.

Data Functions

Read Marker Counter Value	rssifs_readMarkerCounterValue

Class/Panel Name	Function Name
Read Marker Value	rssifs_readMarkerValue
Read Delta Marker Value	rssifs_readDeltaMarkerValue
Read N dB Down Marker Value	rssifs_readNdBDownMarkerValue
Read Noise Marker Value	rssifs_readNoiseMarkerValue
Read Channel Power	rssifs_readChannelPower
Read Occupied Bandwidth	rssifs_readOccupiedBandwidth
Read Time Domain Power	rssifs_readTimeDomainPower

Class/Panel Name	Function Name

Read Delta Marker Value

C Function Prototype

ViStatus rssifs_readDeltaMarkerValue (
    ViSession instrumentHandle,
    ViInt32 deltaMarkerNumber,
    ViReal64* deltaMarkerValue);

Basic Function Prototype

Function rssifs_readDeltaMarkerValue (
    ByVal instrumentHandle As ViSession,
    ByVal deltaMarkerNumber As ViInt32,
    deltaMarkerValue As ViReal64) As ViStatus

Purpose

This function returns delta marker level value relative to the corresponding normal marker position over the trace cache data.

Note

Corresponding marker and delta marker must be enabled!

Parameters List

1. **ViSession instrumentHandle [in]**
   This control accepts the Instrument Handle returned by the Initialize function to select the desired instrument driver session.

   Default Value: None

2. **ViInt32 deltaMarkerNumber [in]**
   Selects delta marker.

   Valid Value: 1 to 2

   Default Value: 1

3. **ViReal64 deltaMarkerValue [out]**
   This control returns delta marker level value relative to the corresponding normal marker position.

Note

Value is returned in current trace unit. See Configure Trace Unit (rssifs_confTraceUnit) function.

Return Value

Returns the status code of this operation.

The meaning of the status code is described in section Error (Status) Codes.

1.6.1.5 Read N dB Down Marker Value

C Function Prototype

ViStatus rssifs_readNdBDownMarkerValue (
    ViSession instrumentHandle,
    ViInt32 markerNumber,
    ViReal64[] spacingValue);

Basic Function Prototype

Function rssifs_readNdBDownMarkerValue (
    ByVal instrumentHandle As ViSession,
    ByVal markerNumber As ViInt32,
    spacingValue As ViReal64) As ViStatus

Purpose

This function queries measurement result of the temporary markers which are n dB below the active reference marker.

Note

Corresponding marker must be enabled!

Parameters List

1. **ViSession instrumentHandle [in]**
   This control accepts the Instrument Handle returned by the Initialize function to select the desired instrument driver session.

   Default Value: None

Class/Panel Name	Function Name
Read Complete Sweep Data	rssifs_readCompleteSweepData
Utility Functions	
Time Out	
Set Time Out	rssifs_setTimeOut
Get Time Out	rssifs_getTimeOut
Flush Error Queue	rssifs_FlushErrorQueue
State Checking	rssifs_errorCheckState
Warning Checking	rssifs_warningCheckState
Reset	rssifs_reset
Self-Test	rssifs_self_test
Error-Query	rssifs_error_query
Error Message	rssifs_error_message
Revision Query	rssifs_revision_query
Close	rssifs_close

Table 0-1: Install program

1.7 Function Tree Layout of the FS300 Spectrum Analyzer

Description

This instrument module provides programming support for the R&S Series300 Spectrum Analyzer. The module is divided into the following functions:

Functions/Classes:

1. **Initialize:**

   This function initializes the instrument and sets it to a default configuration.

2. **Application Functions: (Class)**

   This class contains high-level, test and measurement routines. These examples call instrument driver functions to configure, start, and read from the instrument.

3. **Configuration Functions: (Class)**

   This class of functions configures the instrument by setting acquisition and system configuration parameters.

4. **Action/Status Functions: (Class)**

   This class of functions begins or terminates an acquisition.

5. **Data Functions: (Class)**

   This class of functions transfers data to or from the instrument.

6. **Utility Functions: (Class)**

   This class of functions provides lower level functions to communicate with the instrument, and change instrument parameters. It also provides functions, which allow the user to determine the current status of the instrument.

7. **Close:**

   This function takes the instrument offline.

1.7.1 Initialization

C Function Prototype

```
ViStatus rssifs_init (
          ViRsrc resourceName,
          ViBoolean idQuery,
          ViBoolean resetDevice,
          ViSession* instrumentHandle);
```

Basic Function Prototype

```
Function rssifs_init (
          ByVal resourceName As ViRsrc,
          ByVal idQuery As ViBoolean,
          ByVal resetDevice As ViBoolean,
          instrumentHandle As ViSession) As ViStatus
```

Purpose

This function performs the following initialization actions:

- Opens a session to the specified device using the interface and address specified in the Resource_Name control.
- Performs an identification query on the Instrument.
- Resets the instrument to a known state.
- Sends initialization commands to the instrument.
- Returns an Instrument Handle, which is used to differentiate between different sessions of this instrument driver.
- Each time this function is invoked a Unique Session is opened. It is possible to have more than one session open for the same resource.

Parameters List

1. **ViRsrc resourceName [in]**

   This control specifies the interface and address of the device that is to be initialized (Instrument Descriptor). The exact grammar to be used in this control is shown in the note below.

   Default Value: "USB::0xAAD::0x6::100196"

Note

Based on the Instrument Descriptor, this operation establishes a communication session with a device. The grammar for the Instrument Descriptor is shown below. Optional parameters are shown in square brackets ([]).

```
Interface   Grammar
------------------------------------------------------
USB::manufacturer_ID::model_code::serial_number
```

The USB keyword is used for USB interface. The Serial number is the (Model) Serial Number printed on the instrument box.

The driver also supports logical names. You can pass a logical name instead of instrument descriptor.

Example: "device_1" instead of "USB::0x0aad::0x6::123456".

Logical names can be configured with the SiScan application distributed with the instrument driver package.

2. **ViBoolean idQuery [in]**

   This control specifies if an ID Query is sent to the instrument during the initialization procedure.

   Valid Range:

   - VI_FALSE (0) - Skip Query
   - VI_TRUE (1) - Do Query (Default Value)

Note

Under normal circumstances the ID Query ensures that the instrument initialized is the type supported by this driver. However circumstances may arise where it is undesirable to send an ID Query to the instrument. In those cases, set this control to "Skip Query" and this function will initialize the selected interface, without doing an ID Query.

3. **ViBoolean resetDevice [in]**

   This control specifies if the instrument is to be reset to its power-on settings during the initialization procedure.

   Valid Range:

   - VI_FALSE (0) - Don't Reset
   - VI_TRUE (1) - Reset Device (Default Value)

Note

If you do not want the instrument reset, set this control to "Don't Reset" while initializing the instrument.

4. **ViSession instrumentHandle [out]**

   This control returns an Instrument Handle that is used in all subsequent function calls to differentiate between different sessions of this instrument driver.

Note

Each time this function is invoked a Unique Session is opened. It is possible to have more than one session open for the same resource.

Return Value

Returns the status code of this operation.

The meaning of the status code is described in section Error (Status) Codes.

1.7.2 Application Functions

Description

This class contains high-level test and measurement routines. These examples call other instrument driver functions to configure, start, and read from the instrument.

Functions

Read Spectrum:

- This function shows how to use instrument driver functions. For reasonable measurement is necessary to set up the instrument to the required state.

1.7.2.1 Read Spectrum

C Function Prototype

```
ViStatus rssifs_appReadSpectrum (
      ViSession instrumentHandle,
      ViInt32 measurementMode,
      ViInt32 frequencyParameter,
      ViReal64 startFrequency,
      ViReal64 stopFrequency,
      ViInt32* samplesReturned,
      ViReal64[] traceData);
```

Basic Function Prototype

```
Function rssifs_appReadSpectrum (
      ByVal instrumentHandle As ViSession,
      ByVal measurementMode As ViInt32,
      ByVal frequencyParameter As ViInt32,
      ByVal startFrequency As ViReal64,
      ByVal stopFrequency As ViReal64,
      samplesReturned As ViInt32,
      traceData As ViReal64) As ViStatus
```

Purpose

This function is a simple example how to use instrument driver functions.

This function allows:

- set up start and stop frequencies
- set up continuous or single sweep mode and trigger it
- read out measured trace data

Parameters List

1. **ViSession instrumentHandle [in]**

   This control accepts the Instrument Handle returned by the Initialize function to select the desired instrument driver session.

   Default Value: None

2. **ViInt32 measurementMode [in]**

   This control sets the measurement mode.

   Valid Range:

   - 0 - Set Up Frequencies
   - 1 - Continual Measurement
   - 2 - Single Measurement
   - 3 - Transfer Data Only

   Default Value: 0

Note

- Set Up Frequencies:
  Allows to set up start and stop frequencies.
- Continual Measurement:
  Sets the continual sweeping.
- Single Measurement:
  Sets the single sweeping and immediately sweeps.
- Transfer Data Only:
  Transfers data from the instrument without reconfiguring it.

3. **ViInt32 frequencyParameter [in]**

   Specifies which frequency parameter(s) will be set.

   Valid Range:

   - 0 - Start And Stop (Default Value)
   - 1 - Start Only
   - 2 - Stop Only

4. **ViReal64 startFrequency [in]**

   Sets the start frequency of the analyzer.

   Valid Range: 0.0 Hz to 3.0e9 Hz

   Default Value: 0.0 Hz

Note

Frequency offset is set to 0.0 Hz.

5. **ViReal64 stopFrequency [in]**

   Sets the stop frequency of the analyzer.

   Valid Range: 0.0 Hz to 3.0e9 Hz

   Default Value: 3.0e9 Hz

Note

Frequency offset is set to 0.0 Hz.

6. **ViInt32 samplesReturned [out]**

   Returns the number of retrieved trace data points.

7. **ViReal64[] traceData [out]**

   Returns the trace data.

Note

The array must contain at least 500 elements of ViReal64[] data type.

Transferred data represents rms voltage at the analyzer's input.

Return Value

Returns the status code of this operation.

The meaning of the status code is described in section Error (Status) Codes.

1.7.3 Configuration Functions

Description

This class of functions configures the instrument by setting or retrieving acquisition and system configuration parameters.

Functions

Functions are grouped according the analyzer's subsystems to which they belong.

1.7.3.1 Frequency Settings

Description

This class of functions is used to operate the frequency parameters of the analyzer.

1.7.3.1.1 Configure Start Stop Frequency

C Function Prototype

```
ViStatus rssifs_confStartStopFrq (
      ViSession instrumentHandle,
      ViInt32 frequencyParameter,
      ViReal64 startFrequency,
      ViReal64 stopFrequency);
```

Basic Function Prototype

```
Function rssifs_confStartStopFrq (
      ByVal instrumentHandle As ViSession,
      ByVal frequencyParameter As ViInt32,
      ByVal startFrequency As ViReal64,
      ByVal stopFrequency As ViReal64) As ViStatus
```

Purpose

This function configures start and stop frequencies of the spectrum analyzer.

Note

The frequency of the RF input signal is calculated from the frequency and offset values as follows:

RF input frequency = (frequency - offset)

Where RF input frequency is the frequency of input signal and frequency vs offset are virtual values used in the driver.

Parameters List

1. **ViSession instrumentHandle [in]**

   This control accepts the Instrument Handle returned by the Initialize function to select the desired instrument driver session.

   Default Value: None

2. **ViInt32 frequencyParameter [in]**

   Specifies what frequency parameter is to be configured.

   Valid Range:

   - 0 - Start And Stop
   - 1 - Start Only
   - 2 - Stop Only

   Default Value: 0

3. **ViReal64 startFrequency [in]**

   Sets the start frequency of the analyzer.

   Valid Range (offset = 0.0): 0.0 Hz to 3.0 GHz

   Default Value: 0.0 Hz

4. **ViReal64 stopFrequency [in]**
   Sets the stop frequency of the analyzer.

   Valid Range (offset = 0.0): 0.0 Hz to 3.0 GHz

   Default Value: 3.0e9 Hz

Return Value

Returns the status code of this operation.

The meaning of the status code is described in section Error (Status) Codes.

1.7.3.1.2 Configure Span Center Frequency

C Function Prototype

```
ViStatus rssifs_confSpanCenterFrq (
      ViSession instrumentHandle,
      ViInt32 frequencyParameter,
      ViReal64 frequencySpan,
      ViReal64 centerFrequency);
```

Basic Function Prototype

```
Function rssifs_confSpanCenterFrq (
      ByVal instrumentHandle As ViSession,
      ByVal frequencyParameter As ViInt32,
      ByVal frequencySpan As ViReal64,
      ByVal centerFrequency As ViReal64) As ViStatus
```

Purpose

This function configures frequency span and center frequency of the spectrum analyzer.

Note

When the frequency span is equal to zero the instrument acts as time domain analyzer (Zero span mode). The frequency domain analyzer mode (Sweep mode) is switched on when Start, Stop or Span (> zero) frequency value change applies.

The frequency of the RF input signal is calculated from the frequency and offset values as follows:

RF input frequency = (frequency - offset)

The RF input frequency is the frequency of input signal and frequency and offset are virtual values used in the driver.

Parameters List

1. **ViSession instrumentHandle [in]**

   This control accepts the Instrument Handle returned by the Initialize function to select the desired instrument driver session.

   Default Value: None

2. **ViInt32 frequencyParameter [in]**

   Specifies what frequency parameter is to be configured.

   Valid Range:

   - 0 - Span And Center (Default Value)
   - 1 - Span Only
   - 2 - Center Only

3. **ViReal64 frequencySpan [in]**

   Sets the frequency span of the analyzer.

   Valid Range (offset = 0.0): 0.0 Hz to 3.0 GHz

   Where

   - min = center - span / 2.0
   - max = center + span / 2.0

   Default Value: 3.0e9 Hz

4. **ViReal64 centerFrequency [in]**

   Sets the center frequency of the analyzer.

   Valid Range (offset = 0.0): 0.0 Hz to 3.0 GHz

   Where

   - min = center - span / 2.0
   - max = center + span / 2.0

   Default Value: 1.5e9 Hz

Return Value

Returns the status code of this operation.

The meaning of the status code is described in section Error (Status) Codes.

1.7.3.1.3 Configure Frequency Offset

C Function Prototype

```
ViStatus rssifs_confFreqOffset (
      ViSession instrumentHandle,
      ViReal64 frequencyOffset);
```

Basic Function Prototype

```
Function rssifs_confFreqOffset (
      ByVal instrumentHandle As ViSession,
      ByVal frequencyOffset As ViReal64) As ViStatus
```

Purpose

This function configures the frequency offset of the spectrum analyzer. The frequency offset virtually shifts the resulting frequency.

Note

The frequency of the RF input signal is calculated from the frequency and offset values as follows:

RF input frequency = (frequency - offset)

The RF input frequency is the frequency of input signal and frequency and offset are virtual values used in the driver.

Parameters List

1. **ViSession instrumentHandle [in]**

   This control accepts the Instrument Handle returned by the Initialize function to select the desired instrument driver session.

   Default Value: None

2. **ViReal64 frequencyOffset [in]**

   Sets the frequency offset.

   Valid Range: -100.0 GHz to 100.0 GHz

   Default Value: 0.0 Hz

Return Value

Returns the status code of this operation.

The meaning of the status code is described in section Error (Status) Codes.

1.7.3.1.4 Configure Signal Track

C Function Prototype

```
ViStatus rssifs_confSignalTrack (
      ViSession instrumentHandle,
      ViInt32 signalTrackFunction,
      ViReal64 bandwidth,
      ViReal64 threshold);
```

Basic Function Prototype

```
Function rssifs_confSignalTrack (
      ByVal instrumentHandle As ViSession,
      ByVal signalTrackFunction As ViInt32,
      ByVal bandwidth As ViReal64,
      ByVal threshold As ViReal64) As ViStatus
```

Purpose

This function switches the signal-track On or Off and sets the track bandwidth and threshold to the indicated values. The function is independent of the selected marker.

Note

With signal track activated, the maximum signal is determined after each frequency sweep and the center frequency of this signal is set. With drifting signals the center frequency follows the signal.

Using the function in single sweep mode requires the function Send Trigger and Wait for OPC to be used in order to synchronize subsequent commands correctly to the end of the track action.

This function is only available in the frequency domain (span > 0).

Parameters List

1. **ViSession instrumentHandle [in]**

   This control accepts the Instrument Handle returned by the Initialize function to select the desired instrument driver session.

   Default Value: None

2. **ViInt32 signalTrackFunction [in]**

   This control selects signal track operation or parameter to be set.

   Valid Range:

   - 0 - Signal Track Off
   - 1 - Signal Track On
   - 2 - Bandwidth Value
   - 3 - Threshold Value
   - 4 - Set All

   Default Value: 0 - Signal Track Off

Note

- Signal Track On/Off:
  With signal track activated, the maximum signal is determined after each frequency sweep and the center frequency of this signal is set. With drifting signals the center frequency follows the signal.
- Bandwidth Value:
  This value defines the bandwidth at the center frequency within which the largest signal is searched for in the selected measurement window. Prior to set this value switch signal track to On.
- Threshold Value:
  This value defines the threshold above which the largest signal is searched for in the selected measurement window. Prior to set this value switch signal track to On.

- Set All:
  Sets bandwidth and threshold values together with signal track to On.

3. **ViReal64 bandwidth [in]**

   Sets bandwidth around the center frequency within which the largest signal is searched. By default the bandwidth value is set (= span/10) on activating the function.

   Valid Range: 1000.0 Hz to 3.0e9 Hz

   Default Value: (= span/10) on activating the function

4. **ViReal64 threshold [in]**

   Defines the threshold above which the largest signal is searched for.

   Valid Range (offset = 0.0): -110.0 dBm to 36.0 dBm

   Default Value: -20.0 dBm

Note

Min and max threshold value depends on the reference level and reference level offset. Reference level is calculated from the displayed reference level and offset as follows:

reference level = displayed reference level - offset

Where reference level is the value passed to the device and displayed reference level vs offset are virtual values used in the driver.

Return Value

Returns the status code of this operation.

The meaning of the status code is described in section Error (Status) Codes.

1.7.3.1.5 Low-Level Functions

Description

A class of elementary functions to get (or set) the analyzer's single parameter values.

1.7.3.1.5.1 Get Center Frequency

C Function Prototype

```
ViStatus rssifs_getCenterFrequency (
      ViSession instrumentHandle,
      ViReal64* centerFrequency);
```

Basic Function Prototype

```
Function rssifs_getCenterFrequency (
      ByVal instrumentHandle As ViSession,
      centerFrequency As ViReal64) As ViStatus
```

Purpose

This function gets the center frequency.

Note

Frequency offset applies.

Parameters List

1. **ViSession instrumentHandle [in]**

   This control accepts the Instrument Handle returned by the Initialize function to select the desired instrument driver session.

   Default Value: None

2. **ViReal64 centerFrequency [out]**

   Returns center frequency in Hz.

Return Value

Returns the status code of this operation.

The meaning of the status code is described in section Error (Status) Codes.

1.7.3.1.5.2 Get Frequency Span

C Function Prototype

```
ViStatus rssifs_getFrequencySpan (
      ViSession instrumentHandle,
      ViReal64* frequencySpan);
```

Basic Function Prototype

```
Function rssifs_getFrequencySpan (
      ByVal instrumentHandle As ViSession,
      frequencySpan As ViReal64) As ViStatus
```

Purpose

This function gets the frequency span.

Parameters List

1. **ViSession instrumentHandle [in]**

   This control accepts the Instrument Handle returned by the Initialize function to select the desired instrument driver session.

   Default Value: None

2. **ViReal64 frequencySpan [out]**

   Returns frequency span in Hz.

Return Value

Returns the status code of this operation.

The meaning of the status code is described in section Error (Status) Codes.

1.7.3.1.5.3 Get Frequency Offset

C Function Prototype

```
ViStatus rssifs_getFrequencyOffset (
      ViSession instrumentHandle,
      ViReal64* frequencyOffset);
```

Basic Function Prototype

```
Function rssifs_getFrequencyOffset (
      ByVal instrumentHandle As ViSession,
      frequencyOffset As ViReal64) As ViStatus
```

Purpose

This function gets the frequency offset.

Parameters List

1. **ViSession instrumentHandle [in]**

   This control accepts the Instrument Handle returned by the Initialize function to select the desired instrument driver session.

   Default Value: None

2. **ViReal64 frequencyOffset [out]**

   Returns frequency offset in Hz.

Return Value

Returns the status code of this operation.

The meaning of the status code is described in section Error (Status) Codes.

1.7.3.1.5.4 Get Start Frequency

C Function Prototype

```
ViStatus rssifs_getStartFrequency (
      ViSession instrumentHandle,
      ViReal64* startFrequency);
```

Basic Function Prototype

```
Function rssifs_getStartFrequency (
      ByVal instrumentHandle As ViSession,
      startFrequency As ViReal64) As ViStatus
```

Purpose

This function gets the start frequency.

Note

Frequency offset applies.

Parameters List

1. **ViSession instrumentHandle [in]**

   This control accepts the Instrument Handle returned by the Initialize function to select the desired instrument driver session.

   Default Value: None

2. **ViReal64 startFrequency [out]**

   Returns start frequency in Hz.

Return Value

Returns the status code of this operation.

The meaning of the status code is described in section Error (Status) Codes.

1.7.3.1.5.5 Get Stop Frequency

C Function Prototype

```
ViStatus rssifs_getStopFrequency (
      ViSession instrumentHandle,
      ViReal64* stopFrequency);
```

Basic Function Prototype

```
Function rssifs_getStopFrequency (
      ByVal instrumentHandle As ViSession,
      stopFrequency As ViReal64) As ViStatus
```

Purpose

This function gets the stop frequency.

Note

Frequency offset applies.

Parameters List

1. **ViSession instrumentHandle [in]**

   This control accepts the Instrument Handle returned by the Initialize function to select the desired instrument driver session.

   Default Value: None

2. **ViReal64 stopFrequency [out]**

   Returns stop frequency in Hz.

Return Value

Returns the status code of this operation.

The meaning of the status code is described in section Error (Status) Codes.

1.7.3.1.5.6 Get Signal Track

C Function Prototype

```
ViStatus rssifs_getSignalTrack (
      ViSession instrumentHandle,
      ViInt32* signalTrack,
      ViReal64* bandwidthValue,
      ViReal64* thresholdValue);
```

Basic Function Prototype

```
Function rssifs_getSignalTrack (
      ByVal instrumentHandle As ViSession,
      signalTrack As ViInt32,
      bandwidthValue As ViReal64,
      thresholdValue As ViReal64) As ViStatus
```

Purpose

This function returns parameters of the signal-track (the track bandwidth, threshold and state).

Parameters List

3. **ViSession instrumentHandle [in]**

   This control accepts the Instrument Handle returned by the Initialize function to select the desired instrument driver session.

   Default Value: None

4. **ViInt32 signalTrack [out]**

   Returns whether signal track operation is active or not.

   Valid Range:

   - 0 - Signal Track Off
   - 1 - Signal Track On

5. **ViReal64 bandwidthValue [out]**

   Returns bandwidth around the center frequency within which the largest signal is searched (in Hz).

6. **ViReal64 thresholdValue [out]**

   Returns the threshold above which the largest signal is searched for (in dBm).

Return Value

Returns the status code of this operation.

The meaning of the status code is described in section Error (Status) Codes.

1.7.3.2 Amplitude Settings

Description

This class of functions is used to operate the analyzer's amplitude parameters.

1.7.3.2.1 Configure Reference Level

C Function Prototype

```
ViStatus rssifs_confRefLevel (
      ViSession instrumentHandle,
      ViReal64 referenceLevel);
```

Basic Function Prototype

```
Function rssifs_confRefLevel (
      ByVal instrumentHandle As ViSession,
      ByVal referenceLevel As ViReal64) As ViStatus
```

Purpose

This function configures the reference level.

Note

Reference level is calculated from the displayed reference level and offset as follows:

reference level = displayed reference level - offset

Where reference level is the value passed to the device and displayed reference level vs offset are virtual values used in the driver.

Parameters List

1. **ViSession instrumentHandle [in]**

   This control accepts the Instrument Handle returned by the Initialize function to select the desired instrument driver session.

   Default Value: None

2. **ViReal64 referenceLevel [in]**

   Specifies the reference level.

   Valid Range (offset = 0.0): -110.0 dBm to 36.0 dBm

   Default Value: -20.0 dBm

Return Value

Returns the status code of this operation.

The meaning of the status code is described in section Error (Status) Codes.

1.7.3.2.2 Configure Reference Level Offset

C Function Prototype

```
ViStatus rssifs_confRefLevelOffset (
      ViSession instrumentHandle,
      ViReal64 referenceLevelOffset);
```

Basic Function Prototype

```
Function rssifs_confRefLevelOffset (
      ByVal instrumentHandle As ViSession,
      ByVal referenceLevelOffset As ViReal64) As ViStatus
```

Purpose

This function sets reference level offset.

Note

Reference level is calculated from the displayed reference level and offset as follows:

reference level = displayed reference level – offset

The reference level is the value passed to the device and displayed reference level and offset are virtual values used in the driver.

Parameters List

1. **ViSession instrumentHandle [in]**

   This control accepts the Instrument Handle returned by the Initialize function to select the desired instrument driver session.

   Default Value: None

2. **ViReal64 referenceLevelOffset [in]**

   Defines reference level offset value.

   Valid Range: -100.0 dB to 100.0 dB

   Default Value: 0.0 dB

Return Value

Returns the status code of this operation.

The meaning of the status code is described in section Error (Status) Codes.

1.7.3.2.3 Low-Level Functions

Description A class of elementary functions to get (or set) the analyzer's single parameter values.

1.7.3.2.3.1 Get Reference Level

C Function Prototype

```
ViStatus rssifs_getReferenceLevel (
      ViSession instrumentHandle,
      ViReal64* referenceLevel);
```

Basic Function Prototype

```
Function rssifs_getReferenceLevel (
      ByVal instrumentHandle As ViSession,
      referenceLevel As ViReal64) As ViStatus
```

Purpose This function gets the reference level.

Note Reference level offset applies.

Parameters List

1. **ViSession instrumentHandle [in]**

   This control accepts the Instrument Handle returned by the Initialize function to select the desired instrument driver session.

   Default Value: None

2. **ViReal64 referenceLevel [out]**

   Returns the reference level in dBm.

Return Value Returns the status code of this operation.

The meaning of the status code is described in section Error (Status) Codes.

1.7.3.2.3.2 Get Reference Level Offset

C Function Prototype

```
ViStatus rssifs_getReferenceLevelOffset (
      ViSession instrumentHandle,
      ViReal64* referenceLevelOffset);
```

Basic Function Prototype

```
Function rssifs_getReferenceLevelOffset (
      ByVal instrumentHandle As ViSession,
      referenceLevelOffset As ViReal64) As ViStatus
```

Purpose This function returns reference level offset.

Parameters List

1. **ViSession instrumentHandle [in]**

   This control accepts the Instrument Handle returned by the Initialize function to select the desired instrument driver session.

   Default Value: None

2. **ViReal64 referenceLevelOffset [out]**

   Returns reference level offset value in dB.

Return Value Returns the status code of this operation.

The meaning of the status code is described in section Error (Status) Codes.

1.7.3.3 Input Settings

Description

This class of functions is used to operate the analyzer's input parameters.

1.7.3.3.1 Configure RF Input Attenuation

C Function Prototype

```
ViStatus rssifs_confRFInAtt (
      ViSession instrumentHandle,
      ViReal64 inputAttenuation);
```

Basic Function Prototype

```
Function rssifs_confRFInAtt (
      ByVal instrumentHandle As ViSession,
      ByVal inputAttenuation As ViReal64) As ViStatus
```

Purpose

This function configures the attenuator at the instrument's RF input. It switches RF input attenuator automatically to manual mode.

Parameters List

1. **ViSession instrumentHandle [in]**

   This control accepts the Instrument Handle returned by the Initialize function to select the desired instrument driver session.

   Default Value: None

2. **ViReal64 inputAttenuation [in]**

   This control sets the input attenuation.

   Valid Range: 0.0 to 70.0 dB

   Default Value: 16.0 dB

Note

Attenuation is set in steps. The value passed is then coerced by the instrument to the nearest allowable value. Use function Get RF Input Attenuation (rssifs_getRFInputAttenuation) to obtain the input attenuation value.

Return Value

Returns the status code of this operation.

The meaning of the status code is described in section Error (Status) Codes.

1.7.3.3.2 Configure RF Input Attenuator Auto

C Function Prototype

```
ViStatus rssifs_confRFInAttAuto (
      ViSession instrumentHandle,
      ViInt32 attenuationMode);
```

Basic Function Prototype

```
Function rssifs_confRFInAttAuto (
      ByVal instrumentHandle As ViSession,
      ByVal attenuationMode As ViInt32) As ViStatus
```

Purpose

This function sets analyzer's RF input attenuator to auto mode.

The input attenuation should be set automatically to prevent the R&S FS300's input mixer from being overloaded. There are three RF input modes you can choose from (coupling between reference level and input attenuation) to optimize measurements: Normal, Low Noise and Low Distortion.

Parameters List

1. **ViSession instrumentHandle [in]**

   This control accepts the Instrument Handle returned by the Initialize function to select the desired instrument driver session.

   Default Value: None

2. **ViInt32 attenuationMode [in]**

   Selects the attenuation mode.

   Valid Range:

   - 0 - Auto Normal (Default Value)
   - 1 - Auto Low Noise
   - 2 - Auto Low Distortion

Note

- Auto Normal:
  Normal setting for measurements.
- Auto Low Noise:
  Settings for measurements with low displayed average noise level of the analyzer.
- Auto Low Distortion:
  Setting for measurements with low inherent distortion of the analyzer.

Return Value

Returns the status code of this operation.

The meaning of the status code is described in section Error (Status) Codes.

1.7.3.3.3 Configure RF Input High Sensitivity

C Function Prototype

ViStatus rssifs_confRFInHighSensitivity (
    ViSession instrumentHandle);

Basic Function Prototype

Function rssifs_confRFInHighSensitivity (
    ByVal instrumentHandle As ViSession) As ViStatus

Purpose

This function automatically sets high sensitivity of RF input. It directly affects Input Attenuation and Reference Level parameters:

- Input Attenuation: 0 dB
- Reference Level: -10.0 dBm (offset is not changed)

Note

This function switches RF Input Attenuator automatically to manual mode.

This function switches resolution bandwidth (RBW) to AUTO mode.

This function is an event and therefore has no query and no default value.

Parameters List

1. **ViSession instrumentHandle [in]**

   This control accepts the Instrument Handle returned by the Initialize function to select the desired instrument driver session.

   Default Value: None

Return Value

Returns the status code of this operation.

The meaning of the status code is described in section Error (Status) Codes.

1.7.3.3.4 Low-Level Functions

Description

Class of elementary functions to get (or set) analyzer's parameter value.

1.7.3.3.4.1 Get RF Input Attenuation

C Function Prototype

```
ViStatus rssifs_getRFInputAttenuation (
      ViSession instrumentHandle,
      ViReal64* inputAttenuation);
```

Basic Function Prototype

```
Function rssifs_getRFInputAttenuation (
      ByVal instrumentHandle As ViSession,
      inputAttenuation As ViReal64) As ViStatus
```

Purpose

This function returns the analyzer's RF input attenuation.

Parameters List

1. **ViSession instrumentHandle [in]**

   This control accepts the Instrument Handle returned by the Initialize function to select the desired instrument driver session.

   Default Value: None

2. **ViReal64 inputAttenuation [out]**

   This control returns the analyzer's RF input attenuation value in dB.

Return Value

Returns the status code of this operation.

The meaning of the status code is described in section Error (Status) Codes.

1.7.3.3.4.2 Get RF Input Attenuator Mode

C Function Prototype

```
ViStatus rssifs_getRFInputAttenuatorMode (
      ViSession instrumentHandle,
      ViInt32* attenuationMode);
```

Basic Function Prototype

```
Function rssifs_getRFInputAttenuatorMode (
      ByVal instrumentHandle As ViSession,
      attenuationMode As ViInt32) As ViStatus
```

Purpose

This function returns the analyzer's RF input attenuator auto mode.

Parameters List

1. **ViSession instrumentHandle [in]**

   This control accepts the Instrument Handle returned by the Initialize function to select the desired instrument driver session.

   Default Value: None

2. **ViInt32 attenuationMode [out]**

   Returns the attenuation mode.

   Valid Values:

   - 0 - Auto Normal
   - 1 - Auto Low Noise
   - 2 - Auto Low Distortion

Return Value

Returns the status code of this operation.

The meaning of the status code is described in section Error (Status) Codes.

1.7.3.4 Marker Settings

Description

This class of functions is used to virtually operate with the markers over the trace cache data.

1.7.3.4.1 Configure Marker State

C Function Prototype

```
ViStatus rssifs_confMarkState (
      ViSession instrumentHandle,
      ViInt32 markerNumber,
       ViBoolean markerState);
```

Basic Function Prototype

```
Function rssifs_confMarkState (
      ByVal instrumentHandle As ViSession,
      ByVal markerNumber As ViInt32,
      ByVal markerState As ViBoolean) As ViStatus
```

Purpose

This function is used to enable or disable a selected marker for its operation over the trace cache data.

Parameters List

1. **ViSession instrumentHandle [in]**

   This control accepts the Instrument Handle returned by the Initialize function to select the desired instrument driver session.

   Default Value: None

2. **ViInt32 markerNumber [in]**

   Selects the marker to configure.

   Valid Value: 1 to 2

   Default Value: 1

3. **ViBoolean markerState [in]**

   Enables or disables selected marker.

   Valid Range:

   - VI_FALSE (0) - Off (Default Value)
   - VI_TRUE (1) - On

Return Value

Returns the status code of this operation.

The meaning of the status code is described in section Error (Status) Codes.

1.7.3.4.2 Configure Delta Marker State

C Function Prototype

```
ViStatus rssifs_confDeltaMarkState (
            ViSession instrumentHandle,
            ViInt32 deltaMarkerNumber,
            ViBoolean deltaMarkerState);
```

Basic Function Prototype

```
Function rssifs_confDeltaMarkState (
      ByVal instrumentHandle As ViSession,
      ByVal deltaMarkerNumber As ViInt32,
      ByVal deltaMarkerState As ViBoolean) As ViStatus
```

Purpose

This function is used to enable or disable selected delta marker for its operation over the trace cache data.

Parameters List

4. **ViSession instrumentHandle [in]**

   This control accepts the Instrument Handle returned by the Initialize function to select the desired instrument driver session.

   Default Value: None

5. **ViInt32 deltaMarkerNumber [in]**

   Selects delta marker to configure.

   Valid Value: 1 to 2

   Default Value: 1

6. **ViBoolean deltaMarkerState [in]**

   Enables or disables selected delta marker.

   Valid Range:
   VI_FALSE (0) - Off (Default Value)
   VI_TRUE (1) – On

Return Value

Returns the status code of this operation.

The meaning of the status code is described in section Error (Status) Codes.

1.7.3.4.3 Configure Marker Position

C Function Prototype

```
ViStatus rssifs_confMarkPosition (
      ViSession instrumentHandle,
      ViInt32 markerNumber,
      ViReal64 markerPosition);
```

Basic Function Prototype

```
Function rssifs_confMarkPosition (
      ByVal instrumentHandle As ViSession,
      ByVal markerNumber As ViInt32,
      ByVal markerPosition As ViReal64) As ViStatus
```

Purpose

This function positions the selected marker to the indicated frequency (span > 0) or time (span = 0).

Parameters List

1. **ViSession instrumentHandle [in]**

   This control accepts the Instrument Handle returned by the Initialize function to select the desired instrument driver session.

Default Value: None

2. **ViInt32 markerNumber [in]**

   Selects the marker to configure.

   Valid Value: 1 to 2

   Default Value: 1

3. **ViReal64 markerPosition [in]**

   Sets selected marker to the specified position.

   Valid Range:

   - Frequency (span > 0) (offset = 0.0):
     - FS300/FS315: 0.0 Hz to 3.0e9 Hz
   - Time (span = 0):
     - FS300: 0.0 s to 20.0 s
     - FS315: 0.0 s to 10.0 s

   Default Value: 0.0

Note Frequency offset value applies only when (span > 0).

Return Value

Returns the status code of this operation.

The meaning of the status code is described in section Error (Status) Codes.

1.7.3.4.4 Configure Delta Marker Position

C Function Prototype

```
ViStatus rssifs_confDeltaMarkPosition (
      ViSession instrumentHandle,
      ViInt32 deltaMarkerNumber,
      ViReal64 deltaMarkerPosition);
```

Basic Function Prototype

```
Function rssifs_confDeltaMarkPosition (
        ByVal instrumentHandle As ViSession,
        ByVal deltaMarkerNumber As ViInt32,
        ByVal deltaMarkerPosition As ViReal64) As ViStatus
```

Purpose

This function positions selected delta marker to the indicated frequency (span > 0) or time (span = 0) relative to the corresponding normal marker position.

Parameters List

4. **ViSession instrumentHandle [in]**

   This control accepts the Instrument Handle returned by the Initialize function to select the desired instrument driver session.

   Default Value: None

5. **ViInt32 deltaMarkerNumber [in]**

   Selects delta marker to configure.

   Valid Value: 1 to 2

   Default Value: 1

6. **ViReal64 deltaMarkerPosition [in]**

   Sets selected delta marker to the specified position relative to the corresponding normal marker position.

   Valid Range:

   - Frequency (span > 0) (offset = 0.0):
     FS300/FS315: 0.0 Hz to 3.0e9 Hz
   - Time (span = 0):
     FS300: 0.0 s to 20.0 s
     FS315: 0.0 s to 10.0 s

   Default Value: 0.0

Note

Frequency offset value applies only when (span > 0).

Return Value

Returns the status code of this operation.

The meaning of the status code is described in section Error (Status) Codes.

1.7.3.4.5 Configure Marker Frequency Counter

C Function Prototype

```
ViStatus rssifs_confMarkFreqCnt (
      ViSession instrumentHandle,
      ViInt32 markerNumber,
      ViInt32 counterResolution,
      ViBoolean frequencyMeasurement);
```

Basic Function Prototype

```
Function rssifs_confMarkFreqCnt (
      ByVal instrumentHandle As ViSession,
      ByVal markerNumber As ViInt32,
      ByVal counterResolution As ViInt32,
      ByVal frequencyMeasurement As ViBoolean) As ViStatus
```

Purpose

This function configures the frequency counter.

In order to accurately determine the frequency of a signal, instrument is equipped with a frequency counter which measures the frequency of the RF signal at the intermediate frequency (defined by corresponding marker).

Note

If no marker is enabled when this function is executed, Marker 1 is switched on and set to the center frequency of the trace.

The marker frequency can be changed via Configure Marker Position (rssifs_confMarkPosition) or Marker Search (rssifs_actMarkSearch) functions.

Parameters List

1. **ViSession instrumentHandle [in]**

   This control accepts the Instrument Handle returned by the Initialize function to select the desired instrument driver session.

   Default Value: None

2. **ViInt32 markerNumber [in]**

   Selects the marker to configure.

   Valid Value: 1 to 1

   Default Value: 1

3. **ViInt32 counterResolution [in]**

   Specifies the counter resolution. The time which the frequency counter requires for a measurement is proportional to the selected resolution.

   Valid Range:

   - FS300:
     1 - 1Hz (meas time 1sec)
     2 - 10Hz (meas time 0.1sec)
     3 - 100Hz (meas time 0.1sec)
     4 - 1000Hz (meas time 0.01sec)
   - FS315:
     2 - 10Hz
     3 - 100Hz
     4 - 1000Hz
     5 - 10000Hz

   Default Value: 4

4. **ViBoolean frequencyMeasurement [in]**

   Turns on or off the frequency counter measurement.

   Valid Range:

   - VI_FALSE (0) - Off (Default Value)
   - VI_TRUE (1) - On

Return Value

Returns the status code of this operation.

The meaning of the status code is described in section Error (Status) Codes.

1.7.3.4.6 Configure Marker Peak Excursion

C Function Prototype

```
ViStatus rssifs_confMarkPeakExcursion (
      ViSession instrumentHandle,
      ViReal64 peakExcursion);
```

Basic Function Prototype

```
Function rssifs_confMarkPeakExcursion (
      ByVal instrumentHandle As ViSession,
      ByVal peakExcursion As ViReal64) As ViStatus
```

Purpose

This function specifies the minimum amount a signal level must decrease or increase before it is recognized by the search function as a minimum or maximum.

Parameters List

1. **ViSession instrumentHandle [in]**

   This control accepts the Instrument Handle returned by the Initialize function to select the desired instrument driver session.

   Default Value: None

2. **ViReal64 peakExcursion [in]**

   Specifies the minimum amount a signal level must decrease or increase before it is recognized by the search function as a minimum or maximum.

   Valid Range: 0.0 dB to 80.0 dB

   Default Value: 0.0 dB

Return Value

Returns the status code of this operation.

The meaning of the status code is described in section Error (Status) Codes.

1.7.3.4.7 Configure Marker Search Mode

C Function Prototype

```
ViStatus rssifs_confMarkSearchMode (
      ViSession instrumentHandle,
      ViInt32 searchMode);
```

Basic Function Prototype

```
Function rssifs_confMarkSearchMode (
      ByVal instrumentHandle As ViSession,
      ByVal searchMode As ViInt32) As ViStatus
```

Purpose

This function defines the method used to search over the trace cache data for (min or max) peak occurrence.

Parameters List

1. **ViSession instrumentHandle [in]**

   This control accepts the Instrument Handle returned by the Initialize function to select the desired instrument driver session.

   Default Value: None

2. **ViInt32 searchMode [in]**

   Defines method used to search over the trace cache data for (min or max) peak occurrence.

   Valid Values:

   - 0 - Absolute
   - 1 - Relative

   Default Value: 0

Note

- Absolute:
  In this mode the next (left, right) lower maximum or next (left, right) higher minimum will always be detected.
- Relative:
  In this mode search for the next relative maximum or minimum right or left of the current marker position irrespective of the current signal amplitude is provided. Relative maximum is understood to mean a decrease of the signal amplitude by a defined value - i.e. the peak excursion - right and left of the amplitude peak.

Return Value

Returns the status code of this operation.

The meaning of the status code is described in section Error (Status) Codes.

1.7.3.4.8 Marker Search

C Function Prototype

```
ViStatus rssifs_actMarkSearch (
      ViSession instrumentHandle,
      ViInt32 markerNumber,
       ViInt32 search);
```

Basic Function Prototype

```
Function rssifs_actMarkSearch (
      ByVal instrumentHandle As ViSession,
      ByVal markerNumber As ViInt32,
      ByVal search As ViInt32) As ViStatus
```

Purpose

This function is used to search over the trace cache data for (min or max) peak occurrence.

If no extremum value is found on the trace cache data, an execution error (RSSIFS_ERROR_EXECUTION_ERROR (0xBFFC09FE)) is produced.

This function uses previously collected trace data stored in the trace data cache. Measurement can be executed by call of Send Trigger (rssifs_actSendTrg) or Send Trigger and Wait for OPC (rssifs_actSendTrgWopc) function.

Note

Corresponding marker must be enabled!

Execution of this function is influenced by the settings of functions Configure Marker Peak Excursion (rssifs_confMarkPeakExcursion) and Configure Marker Search Mode (rssifs_confMarkSearchMode).

Parameters List

1. ViSession instrumentHandle [in]

   This control accepts the Instrument Handle returned by the Initialize functionto select the desired instrument driver session.

   Default Value: None

2. **ViInt32 markerNumber [in]**

   Selects the marker to configure.

   Valid Value: 1 to 2

   Default Value: 1

3. **ViInt32 search [in]**

   Defines method of the peak searching.

   Valid Range:

   - 0 - Peak Maximum
   - 1 - Next Maximum
   - 2 - Left Maximum
   - 3 - Right Maximum
   - 4 - Peak Minimum
   - 5 - Next Minimum
   - 6 - Left Minimum
   - 7 - Right Minimum

   Default Value: 0

Note

- Peak Minimum (Maximum):
  Sets the active marker to the peak of the trace.
- Next Peak Minimum (Maximum):
  Sets the active marker to the next lower (upper) maximum of the current marker position on the selected trace.
- Left (Right) Peak Minimum (Maximum):
  Sets the active marker to the next lower (upper) maximum left (right) of the current marker position on the selected trace.

Return Value

Returns the status code of this operation.

The meaning of the status code is described in section Error (Status) Codes.

1.7.3.4.9 Marker Search N dB Down

C Function Prototype

```
ViStatus rssifs_confMarkerSearchNdBDown (
    ViSession instrumentHandle,
    ViInt32 markerNumber,
    ViReal64 ndBDownValue,
    ViBoolean ndBDown);
```

Basic Function Prototype

```
Function rssifs_confMarkerSearchNdBDown (
    ByVal instrumentHandle As ViSession,
    ByVal markerNumber As ViInt32,
    ByVal ndBDownValue As ViReal64,
    ByVal ndBDown As ViBoolean) As ViStatus
```

Purpose

This function configures the measurement of the temporary markers which are n dB below the active reference marker.

Parameters List

4. **ViSession instrumentHandle [in]**

   This control accepts the Instrument Handle returned by the Initialize functionto select the desired instrument driver session.

   Default Value: None

5. **ViInt32 markerNumber [in]**

   Selects the marker to configure.

   Valid Value: 1 to 2

   Default Value: 1

6. **ViReal64 ndBDownValue [in]**

   Enters the N dB down value.

   Valid Range: 0.0 dB to 100.0 dB

   Default Value: 3.0 dB

7. **ViBoolean ndBDown [in]**

   Activates and deactivates temporary markers which are located n dB below the active reference marker.

   Valid Range:
   - VI_FALSE (0) - Off (Default Value)
   - VI_TRUE (1) - On

Return Value

Returns the status code of this operation.

The meaning of the status code is described in section Error (Status) Codes.

1.7.3.4.10 Low-Level Functions

Description

A class of elementary functions to get (or set) the analyzer's single parameter values.

1.7.3.4.10.1 Get Marker State

C Function Prototype

```
ViStatus rssifs_getMarkerState (
     ViSession instrumentHandle,
     ViInt32 markerNumber,
     ViBoolean* markerState);
```

Basic Function Prototype

```
Function rssifs_getMarkerState (
     ByVal instrumentHandle As ViSession,
     ByVal markerNumber As ViInt32,
     markerState As ViBoolean) As ViStatus
```

Purpose

This function returns the state of selected marker.

Parameters List

1. **ViSession instrumentHandle [in]**

   This control accepts the Instrument Handle returned by the Initialize function to select the desired instrument driver session.

   Default Value: None

2. **ViInt32 markerNumber [in]**

   Selects the marker.

   Valid Value: 1 to 2

   Default Value: 1

3. **ViBoolean markerState [out]**

   Returns the state of selected marker.

   Valid Values:

   - VI_FALSE (0) - Off
   - VI_TRUE (1) - On

Return Value

Returns the status code of this operation.

The meaning of the status code is described in section Error (Status) Codes.

1.7.3.4.10.2 Get Delta Marker State

C Function Prototype

```
ViStatus rssifs_getDeltaMarkerState (
      ViSession instrumentHandle,
      ViInt32 deltaMarkerNumber,
      ViBoolean* deltaMarkerState);
```

Basic Function Prototype

```
Function rssifs_getDeltaMarkerState (
      ByVal instrumentHandle As ViSession,
      ByVal deltaMarkerNumber As ViInt32,
      deltaMarkerState As ViBoolean) As ViStatus
```

Purpose

This function returns the state of selected delta marker.

Parameters List

4. **ViSession instrumentHandle [in]**

   This control accepts the Instrument Handle returned by the Initialize function to select the desired instrument driver session.

   Default Value: None

5. **ViInt32 deltaMarkerNumber [in]**

   Selects delta marker to configure.

   Valid Value: 1 to 2

   Default Value: 1

6. **ViBoolean deltaMarkerState [out]**

   Returns the state of selected delta marker.

   Valid Values:
   - VI_FALSE (0) – Off
   - VI_TRUE (1) - On

Return Value

Returns the status code of this operation.

The meaning of the status code is described in section Error (Status) Codes.

1.7.3.4.10.3 Get Marker Position

C Function Prototype

```
ViStatus rssifs_getMarkerPosition (
      ViSession instrumentHandle,
      ViInt32 markerNumber,
      ViReal64* markerPosition);
```

Basic Function Prototype

```
Function rssifs_getMarkerPosition (
      ByVal instrumentHandle As ViSession,
      ByVal markerNumber As ViInt32,
      markerPosition As ViReal64) As ViStatus
```

Purpose

This function returns position of the corresponding marker.

Parameters List

1. **ViSession instrumentHandle [in]**

   This control accepts the Instrument Handle returned by the Initialize function to select the desired instrument driver session.

   Default Value: None

2. **ViInt32 markerNumber [in]**

   Selects the marker.

   Valid Value: 1 to 2

Default Value: 1

3. **ViReal64 markerPosition [out]**

   Returns position of the corresponding marker in Hz (span > 0) or Seconds (span = 0).

Return Value

Returns the status code of this operation.

The meaning of the status code is described in section Error (Status) Codes.

1.7.3.4.10.4 Get Delta Marker Position

C Function Prototype

```
ViStatus rssifs_getDeltaMarkPosition (
      ViSession instrumentHandle,
      ViInt32 deltaMarkerNumber,
      ViReal64* deltaMarkerPosition);
```

Basic Function Prototype

```
Function rssifs_getDeltaMarkPosition (
      ByVal instrumentHandle As ViSession,
      ByVal deltaMarkerNumber As ViInt32,
      deltaMarkerPosition As ViReal64) As ViStatus
```

Purpose

This function returns positions of selected delta marker relative to the corresponding normal marker position.

Parameters List

4. **ViSession instrumentHandle [in]**

   This control accepts the Instrument Handle returned by the Initialize function to select the desired instrument driver session.

   Default Value: None

5. **ViInt32 deltaMarkerNumber [in]**

   Selects delta marker to configure.

   Valid Value: 1 to 2

   Default Value: 1

6. **ViReal64 deltaMarkerPosition [out]**

   Returns position of selected delta marker relative to the corresponding normal marker position. Value is represented as frequency (span > 0) or time (span = 0).

Return Value

Returns the status code of this operation.

The meaning of the status code is described in section Error (Status) Codes.

1.7.3.4.10.5 Get Marker Frequency Counter State

C Function Prototype

```
ViStatus rssifs_getMarkerFreqCounterState (
      ViSession instrumentHandle,
      ViInt32 markerNumber,
       ViBoolean* counterState);
```

Basic Function Prototype

```
Function rssifs_getMarkerFreqCounterState (
      ByVal instrumentHandle As ViSession,
      ByVal markerNumber As ViInt32,
      counterState As ViBoolean) As ViStatus
```

Purpose

This function returns the frequency counter state.

Parameters List

1. **ViSession instrumentHandle [in]**

   This control accepts the Instrument Handle returned by the Initialize function to select the desired instrument driver session.

   Default Value: None

2. **ViInt32 markerNumber [in]**

   Selects the marker to configure.

   Valid Value: 1 to 1

   Default Value: 1

3. **ViBoolean counterState [out]**

   Returns frequency counter state.

   Valid Values:

   - VI_FALSE (0) - Off
   - VI_TRUE (1) - On

Return Value

Returns the status code of this operation.

The meaning of the status code is described in section Error (Status) Codes.

1.7.3.4.10.6 Get Marker Frequency Counter Resolution

C Function Prototype

```
ViStatus rssifs_getMarkerFreqCounterResolution (
      ViSession instrumentHandle,
      ViInt32 markerNumber,
      ViInt32* counterResolution);
```

Basic Function Prototype

```
Function rssifs_getMarkerFreqCounterResolution (
      ByVal instrumentHandle As ViSession,
      ByVal markerNumber As ViInt32,
      counterResolution As ViInt32) As ViStatus
```

Purpose

This function returns the frequency counter resolution.

Parameters List

1. **ViSession instrumentHandle [in]**

   This control accepts the Instrument Handle returned by the Initialize function to select the desired instrument driver session.

   Default Value: None

2. **ViInt32 markerNumber [in]**

   Selects the marker to configure.

   Valid Value: 1 to 1

   Default Value: 1

3. **ViInt32 counterResolution [out]**

   Returns the counter resolution. The time which the frequency counter requires for a measurement is proportional to the selected resolution.

   Valid Range:

▪ 1 - 1Hz	(meas time 1sec)
▪ 2 - 10Hz	(meas time 0.1sec)
▪ 3 - 100Hz	(meas time 0.1sec)
▪ 4 - 1000Hz	(meas time 0.01sec)

Return Value

Returns the status code of this operation.

The meaning of the status code is described in section Error (Status) Codes.

1.7.3.4.10.7 Get Marker Peak Excursion

C Function Prototype

```
ViStatus rssifs_getMarkerPeakExcursion (
      ViSession instrumentHandle,
      ViReal64* peakExcursion);
```

Basic Function Prototype

```
Function rssifs_getMarkerPeakExcursion (
      ByVal instrumentHandle As ViSession,
      peakExcursion As ViReal64) As ViStatus
```

Purpose

This function returns a specified minimum amount by which a signal level must Decrease or increase before it is recognized by the search function as a minimum or maximum.

Parameters List

1. **ViSession instrumentHandle [in]**

   This control accepts the Instrument Handle returned by the Initialize function to select the desired instrument driver session.

   Default Value: None

2. **ViReal64 peakExcursion [out]**

   Returns the minimum amount by which signal level must decrease or increase before it is recognized by the search function as a minimum or maximum (in dB).

Return Value

Returns the status code of this operation.

The meaning of the status code is described in section Error (Status) Codes.

1.7.3.4.10.8 Get Marker Search Mode

C Function Prototype

```
ViStatus rssifs_getMarkSearchMode (
      ViSession instrumentHandle,
      ViInt32* searchMode);
```

Basic Function Prototype

```
Function rssifs_getMarkSearchMode (
      ByVal instrumentHandle As ViSession,
      searchMode As ViInt32) As ViStatus
```

Purpose

This function returns a method defined to search over the trace cache data for (min or max) peak occurrence.

Parameters List

1. **ViSession instrumentHandle [in]**

   This control accepts the Instrument Handle returned by the Initialize function to select the desired instrument driver session.

Default Value: None

2. **ViInt32 searchMode [out]**

   Returns method defined to search over the trace cache data for (min or max) peak occurrence.

   Valid Values:

   - 0 - Absolute
   - 1 - Relative

Return Value

Returns the status code of this operation.

The meaning of the status code is described in section Error (Status) Codes.

1.7.3.4.10.9 Get Marker Search N dB Down

C Function Prototype

```
ViStatus rssifs_getMarkerSearchNdBDown (
      ViSession instrumentHandle,
      ViInt32 markerNumber,
      ViReal64* ndBDownValue,
      ViBoolean* ndBDown);
```

Basic Function Prototype

```
Function rssifs_getMarkerSearchNdBDown (
      ByVal instrumentHandle As ViSession,
      ByVal markerNumber As ViInt32,
      ndBDownValue As ViReal64,
      ndBDown As ViBoolean) As ViStatus
```

Purpose

This function returns measurement settings of the temporary markers which are n dB below the active reference marker.

Parameters List

3. **ViSession instrumentHandle [in]**

   This control accepts the Instrument Handle returned by the Initialize function to select the desired instrument driver session.

   Default Value: None

4. **ViInt32 markerNumber [in]**

   Selects the marker to configure.

   Valid Value: 1 to 2

   Default Value: 1

5. **ViReal64 ndBDownValue [out]**

   Returns the N dB down value.

6. **ViBoolean ndBDown [out]**

   Returns whether N dB Down marker measurement is enabled or disabled.

   Valid Range:

   - VI_FALSE (0) – Off
   - VI_TRUE (1) – On

Return Value

Returns the status code of this operation.

The meaning of the status code is described in section Error (Status) Codes.

1.7.3.5 Trace Settings

Description

This class of functions is used to define the type of the evaluation of trace cache data as a whole.

1.7.3.5.1 Configure Trace Mode

C Function Prototype

```
ViStatus rssifs_confTraceMode (
      ViSession instrumentHandle,
      ViInt32 traceMode);
```

Basic Function Prototype

```
Function rssifs_confTraceMode (
      ByVal instrumentHandle As ViSession,
      ByVal traceMode As ViInt32) As ViStatus
```

Purpose

This function defines the type of the evaluation of trace cache data as a whole. Trace can be overwritten in each measurement (CLEAR/WRITE mode), averaged over several measurements (AVERAGE mode) and maximum or minimum value can be determined from several measurements (MAX HOLD or MIN HOLD).

Note

The value of a signal can be determined over several sweeps.

To clear trace cache and restart selected trace mode call this function again.

Parameters List

1. **ViSession instrumentHandle [in]**

   This control accepts the Instrument Handle returned by the Initialize function to select the desired instrument driver session.

   Default Value: None

2. **ViInt32 traceMode [in]**

   Defines the type of the evaluation of the trace cache.

   Valid Range:

▪ RSSIFS_TRACE_MODE_CLEAR_WRITE	(0) - Clear/Write
▪ RSSIFS_TRACE_MODE_VIEW	(1) - View
▪ RSSIFS_TRACE_MODE_AVERAGE	(2) - Average
▪ RSSIFS_TRACE_MODE_MAX_HOLD	(3) - Max Hold
▪ RSSIFS_TRACE_MODE_MIN_HOLD	(4) - Min Hold

   Default Value: RSSIFS_TRACE_MODE_CLEAR_WRITE (0)

   - Clear/Write:
   Activates the overwrite mode for the collected measured values, ie the trace is overwritten by each sweep read out from the instrument.

   - View:
   Freezes the current contents of the trace cache.

   - Average:
   Averaging is carried out over the values derived from the measurement samples.Several measured values may be combined in a averaged trace data. When selected, the trace cache is always cleared.

   - Max Hold:
   Activates the sweep result in the trace cache only if the new value is greater than the previous one. When selected, the trace cache is always cleared.

   - Min Hold:
   Activates the sweep result in the trace cache only if the new value is smaller than the previous one. When selected, the trace cache is always cleared.

Return Value

Returns the status code of this operation.

The meaning of the status code is described in section Error (Status) Codes.

1.7.3.5.2 Configure Trace Detector

C Function Prototype

```
ViStatus rssifs_confTraceDetector (
      ViSession instrumentHandle,
      ViInt32 traceDetector);
```

Basic Function Prototype

```
Function rssifs_confTraceDetector (
      ByVal instrumentHandle As ViSession,
      ByVal traceDetector As ViInt32) As ViStatus
```

Purpose

This function controls the recording of trace values via the type of detector.

Note

This function is available for FS315 only.

Parameters List

3. **ViSession instrumentHandle [in]**

   This control accepts the Instrument Handle returned by the Initialize function to select the desired instrument driver session.

   Default Value: None

4. **ViInt32 traceDetector [in]**

   Defines the trace detector used.

   Valid Range:

   - RSSIFS_TRACE_DETECTOR_MIN_PEAK (1) - Min Peak
   - RSSIFS_TRACE_DETECTOR_MAX_PEAK (2) - Max Peak
   - RSSIFS_TRACE_DETECTOR_SAMPLE (3) – Sample
   - RSSIFS_TRACE_DETECTOR_RMS (4) – RMS
   - RSSIFS_TRACE_DETECTOR_AVG (5) - Average

   Default Value: RSSIFS_TRACE_DETECTOR_MAX_PEAK (2)

Note

- Min Peak:
  The min peak detector selects from the samples allocated to a pixel the one with the minimum value.

- Max Peak:
  The max peak detector selects from the samples allocated to a pixel the one with the maximum value.

- Sample:
  The sample detector samples the IF envelope for each pixel of the trace to be displayed only once.

- RMS:
  The RMS (root mean square) detector calculates the power for each pixel of the displayed trace from the samples allocated to a pixel. The result corresponds to the signal power within the span represented by the pixel.

- Average:
  The average detector calculates the linear average for each pixel of the displayed trace from the samples allocated to a pixel.

Return Value

Returns the status code of this operation.

The meaning of the status code is described in section Error (Status) Codes.

1.7.3.5.3 Configure Trace Unit

C Function Prototype

```
ViStatus rssifs_confTraceUnit (
      ViSession instrumentHandle,
      ViInt32 traceUnit);
```

Basic Function Prototype

```
Function rssifs_confTraceUnit (
      ByVal instrumentHandle As ViSession,
      ByVal traceUnit As ViInt32) As ViStatus
```

Purpose

This function defines trace data unit.

Parameters List

5. **ViSession instrumentHandle [in]**

This control accepts the Instrument Handle returned by the Initialize function to select the desired instrument driver session.

Default Value: None

6. **ViInt32 traceUnit [in]**

Defines the trace data unit.

Valid Range:

- RSSIFS_TRACE_UNIT_VOLT (0) – Volt
- RSSIFS_TRACE_UNIT_WATT (1) - Watt
- RSSIFS_TRACE_UNIT_DBM (2) - dBm
- RSSIFS_TRACE_UNIT_DBMV (3) - dBmV
- RSSIFS_TRACE_UNIT_DBUV (4) - dBuV
- RSSIFS_TRACE_UNIT_DBUA (5) - dBuA

Default Value: RSSIFS_TRACE_UNIT_VOLT (0)

Return Value

Returns the status code of this operation.

The meaning of the status code is described in section Error (Status) Codes.

1.7.3.5.4 Low-Level Functions

Description

A class of elementary functions to get (or set) the analyzer's single parameter values.

1.7.3.5.4.1 Get Trace Mode

C Function Prototype

```
ViStatus rssifs_getTraceMode (
      ViSession instrumentHandle,
      ViInt32* traceMode);
```

Basic Function Prototype

```
Function rssifs_getTraceMode (
      ByVal instrumentHandle As ViSession,
      traceMode As ViInt32) As ViStatus
```

Purpose

This function gets the type of the evaluation of trace cache data.

Parameters List

1. **ViSession instrumentHandle [in]**

   This control accepts the Instrument Handle returned by the Initialize function to select the desired instrument driver session.

   Default Value: None

2. **ViInt32 traceMode [out]**

   Defines the type of the evaluation of the traces.

   Valid Range:

   - RSSIFS_TRACE_MODE_CLEAR_WRITE (0) - Clear/Write
   - RSSIFS_TRACE_MODE_VIEW (1) – View
   - RSSIFS_TRACE_MODE_AVERAGE - (2) Average
   - RSSIFS_TRACE_MODE_MAX_HOLD (3) - Max Hold
   - RSSIFS_TRACE_MODE_MIN_HOLD (4) - Min Hold

Return Value

Returns the status code of this operation.

The meaning of the status code is described in section Error (Status) Codes.

1.7.3.5.4.2 Get Trace Detector

C Function Prototype

```
ViStatus rssifs_getTraceDetector (
      ViSession instrumentHandle,
      ViInt32* traceDetector);
```

Basic Function Prototype

```
Function rssifs_getTraceDetector (
      ByVal instrumentHandle As ViSession,
      traceDetector As ViInt32) As ViStatus
```

Purpose

This function returns the trace detector used.

Note

This function is available for FS315 only.

Parameters List

3. **ViSession instrumentHandle [in]**

   This control accepts the Instrument Handle returned by the Initialize function to select the desired instrument driver session.

   Default Value: None

4. **ViInt32 traceDetector [out]**

   Returns the trace detector used.

Valid Range:

- RSSIFS_TRACE_DETECTOR_MIN_PEAK (1) - Min Peak
- RSSIFS_TRACE_DETECTOR_MAX_PEAK (2) - Max Peak
- RSSIFS_TRACE_DETECTOR_SAMPLE (3) – Sample
- RSSIFS_TRACE_DETECTOR_RMS (4) – RMS
- RSSIFS_TRACE_DETECTOR_AVG (5) - Average

Return Value

Returns the status code of this operation.

The meaning of the status code is described in section Error (Status) Codes.

1.7.3.5.4.3 Get Trace Unit

C Function Prototype

```
ViStatus rssifs_getTraceUnit (
      ViSession instrumentHandle,
      ViInt32* traceUnit);
```

Basic Function Prototype

```
Function rssifs_getTraceUnit (
      ByVal instrumentHandle As ViSession,
      traceUnit As ViInt32) As ViStatus
```

Purpose

This function returns current trace data unit.

Parameters List

5. **ViSession instrumentHandle [in]**

   This control accepts the Instrument Handle returned by the Initialize function to select the desired instrument driver session.

   Default Value: None

6. **ViInt32 traceUnit [out]**

   Returns current trace data unit.

   Valid Range:

   - RSSIFS_TRACE_UNIT_VOLT (0) - Volt
   - RSSIFS_TRACE_UNIT_WATT (1) - Watt
   - RSSIFS_TRACE_UNIT_DBM (2) - dBm
   - RSSIFS_TRACE_UNIT_DBMV (3) - dBmV
   - RSSIFS_TRACE_UNIT_DBUV (4) - dBuV
   - RSSIFS_TRACE_UNIT_DBUA (5) - dBuA

Return Value

Returns the status code of this operation.

The meaning of the status code is described in section Error (Status) Codes.

1.7.3.6 Demodulator Settings

Description

This class of functions is used to operate demodulator parameters.

1.7.3.6.1 Configure Demodulator

C Function Prototype

```
ViStatus rssifs_confDemodulator (
      ViSession instrumentHandle,
      ViBoolean demodulatorState,
      ViInt32 demodulatorType);
```

Basic Function Prototype

```
Function rssifs_confDemodulator (
      ByVal instrumentHandle As ViSession,
      ByVal demodulatorState As ViBoolean,
      ByVal demodulatorType As ViInt32) As ViStatus
```

Purpose

This function controls the state and type of demodulator.

Note

This function is available for FS315 only.

Parameters List

7. **ViSession instrumentHandle [in]**

   This control accepts the Instrument Handle returned by the Initialize function to select the desired instrument driver session.

   Default Value: None

8. **ViBoolean demodulatorState [in]**

   Turns demodulator on or off.

   Valid Range:
   - VI_FALSE (0) - Off (Default Value)
   - VI_TRUE (1) - On

9. **ViInt32 demodulatorType [in]**

   Specifies demodulator type.

   Valid Values:
   - 0 - AM
   - 1 - Reserved
   - 2 - FM

   Default Value: 0

Return Value

Returns the status code of this operation.

The meaning of the status code is described in section Error (Status) Codes.

1.7.3.6.2 Configure Demodulator Volume

C Function Prototype

```
ViStatus rssifs_confDemodulatorVolume (
      ViSession instrumentHandle,
      ViReal64 volume);
```

Basic Function Prototype

```
Function rssifs_confDemodulatorVolume (
      ByVal instrumentHandle As ViSession,
      ByVal volume As ViReal64) As ViStatus
```

Purpose

This function sets demodulator AF output volume.

Note

This function is available for FS315 only.

Parameters List

3. **ViSession instrumentHandle [in]**

   This control accepts the Instrument Handle returned by the Initialize function to select the desired instrument driver session.

   Default Value: None

4. **ViReal64 volume [in]**

   Specifies the AF output volume.

   Valid Range: 0.0 % to 100.0 %

   Default Value: 0.0 %

Return Value

Returns the status code of this operation.

The meaning of the status code is described in section Error (Status) Codes.

1.7.3.6.3 Configure Demodulator Appearance

C Function Prototype

```
ViStatus rssifs_confDemodulatorAppearance (
      ViSession instrumentHandle,
      ViInt32 demodulatorTime,
       ViInt32 demodulatorDisplay);
```

Basic Function Prototype

```
Function rssifs_confDemodulatorAppearance (
      ByVal instrumentHandle As ViSession,
      ByVal demodulatorTime As ViInt32,
      ByVal demodulatorDisplay As ViInt32) As ViStatus
```

Purpose

This function configures appearance of demodulator parameters (time and display).

Note

This function is available for FS315 only.

Parameters List

4. **ViSession instrumentHandle [in]**

   This control accepts the Instrument Handle returned by the Initialize function to select the desired instrument driver session.

   Default Value: None

5. **ViInt32 demodulatorTime [in]**

   Specifies demodulator time type.

   Valid Values:

- 0 - Periodic
- 1 - Continuous

Default Value: 0

6. ViInt32 demodulatorDisplay [in]

Specifies demodulator display type.

Valid Values:

- 0 - Carrier
- 1 - Waveform

Default Value: 0

Return Value

Returns the status code of this operation.

The meaning of the status code is described in section Error (Status) Codes.

1.7.3.6.4 Low-Level Functions

Description

Class of elementary functions to get (or set) demodulator parameters value.

1.7.3.6.4.1 Get Demodulator State

C Function Prototype

```
ViStatus rssifs_getDemodulatorState (
      ViSession instrumentHandle,
      ViBoolean* demodulatorState);
```

Basic Function Prototype

```
Function rssifs_getDemodulatorState (
      ByVal instrumentHandle As ViSession,
      demodulatorState As ViBoolean) As ViStatus
```

Purpose

This function returns demodulator state.

Note

This function is available for FS315 only.

Parameters List

3. **ViSession instrumentHandle [in]**

   This control accepts the Instrument Handle returned by the Initialize function to select the desired instrument driver session.

   Default Value: None

4. **ViBoolean demodulatorState [out]**

   Returns demodulator state.

   Valid Values:
   - VI_FALSE (0) - Off
   - VI_TRUE (1) - On

Return Value

Returns the status code of this operation.

The meaning of the status code is described in section Error (Status) Codes.

1.7.3.6.4.2 Get Demodulator Type

C Function Prototype

```
ViStatus rssifs_getDemodulatorType (
      ViSession instrumentHandle,
      ViInt32* demodulatorType);
```

Basic Function Prototype

```
Function rssifs_getDemodulatorType (
      ByVal instrumentHandle As ViSession,
      demodulatorType As ViInt32) As ViStatus
```

Purpose

This function returns demodulator type.

Note

This function is available for FS315 only.

Parameters List

3. **ViSession instrumentHandle [in]**

   This control accepts the Instrument Handle returned by the Initialize function to select the desired instrument driver session.

   Default Value: None

4. **ViInt32 demodulatorType [out]**

   Returns demodulator type.

   Valid Values:
   - 0 - AM
   - 1 - Reserved
   - 2 - FM

Return Value

Returns the status code of this operation.

The meaning of the status code is described in section Error (Status) Codes.

1.7.3.6.4.3 Get Demodulator Volume

C Function Prototype

ViStatus rssifs_getDemodulatorVolume (
    ViSession instrumentHandle,
    ViReal64* demodulatorVolume);

Basic Function Prototype

Function rssifs_getDemodulatorVolume (
    ByVal instrumentHandle As ViSession,
    demodulatorVolume As ViReal64) As ViStatus

Purpose

This function returns demodulator AF output volume.

Note

This function is available for FS315 only.

Parameters List

3. **ViSession instrumentHandle [in]**
   This control accepts the Instrument Handle returned by the Initialize function to select the desired instrument driver session.

   Default Value: None

4. **ViReal64 demodulatorVolume [out]**
   Returns the AF output volume in percent.

1.7.3.6.4.4 Get Demodulator Time

C Function Prototype

ViStatus rssifs_getDemodulatorTime (
    ViSession instrumentHandle,
    ViInt32* demodulatorTime);

Basic Function Prototype

Function rssifs_getDemodulatorTime (
    ByVal instrumentHandle As ViSession,
    demodulatorTime As ViInt32) As ViStatus

Purpose

This function returns demodulator time type.

Note

This function is available for FS315 only.

Parameters List

3. **ViSession instrumentHandle [in]**
   This control accepts the Instrument Handle returned by the Initialize function to select the desired instrument driver session.

   Default Value: None

4. **ViInt32 demodulatorTime [out]**
   Returns demodulator time type.

   Valid Values:
   - 0 - Periodic
   - 1 - Continuous

Return Value

Returns the status code of this operation.

The meaning of the status code is described in section Error (Status) Codes.

1.7.3.6.4.5 Get Demodulator Display

C Function Prototype

```
ViStatus rssifs_getDemodulatorDisplay (
      ViSession instrumentHandle,
      ViInt32* demodulatorDisplay);
```

Basic Function Prototype

```
Function rssifs_getDemodulatorDisplay (
      ByVal instrumentHandle As ViSession,
      demodulatorDisplay As ViInt32) As ViStatus
```

Purpose

This function returns demodulator display type.

Note

This function is available for FS315 only.

Parameters List

3. **ViSession instrumentHandle [in]**

   This control accepts the Instrument Handle returned by the Initialize function to select the desired instrument driver session.

   Default Value: None

4. **ViInt32 demodulatorDisplay [out]**

   Returns demodulator display type.

   Valid Values:
   - 0 - Carrier
   - 1 - Waveform

Return Value

Returns the status code of this operation.

The meaning of the status code is described in section Error (Status) Codes.

1.7.3.7 Tracking Generator Settings

Description This class of functions is used to operate tracking generator parameters.

1.7.3.7.1 Configure Tracking Generator

C Function Prototype

```
ViStatus rssifs_confTrackingGenerator (
      ViSession instrumentHandle,
      ViInt32 trackingGeneratorState);
```

Basic Function Prototype

```
Function rssifs_confTrackingGenerator (
      ByVal instrumentHandle As ViSession,
      ByVal trackingGeneratorState As ViInt32) As ViStatus
```

Purpose This function controls the state of tracking generator.

Note This function is available for FS315 only.

Parameters List

3. **ViSession instrumentHandle [in]**
   This control accepts the Instrument Handle returned by the Initialize function to select the desired instrument driver session.

   Default Value: None

4. **ViInt32 trackingGeneratorState [in]**
   Turns tracking generator state.

   Valid Values:
   - 0 - Off
   - 1 - Tracking
   - 2 - Discrete

   Default Value: 0

Return Value Returns the status code of this operation.

The meaning of the status code is described in section Error (Status) Codes.

1.7.3.7.2 Configure Tracking Generator Level

C Function Prototype

```
ViStatus rssifs_confTrackingGeneratorLevel (
      ViSession instrumentHandle,
      ViReal64 level);
```

Basic Function Prototype

```
Function rssifs_confTrackingGeneratorLevel (
      ByVal instrumentHandle As ViSession,
      ByVal level As ViReal64) As ViStatus
```

Purpose This function controls the level of tracking generator.

Note This function is available for FS315 only.

Parameters List

3. **ViSession instrumentHandle [in]**
   This control accepts the Instrument Handle returned by the Initialize function to select the desired instrument driver session.

   Default Value: None

4. **ViReal64 level [in]**
   Defines the level of the tracking generator.

Valid Range: -50.0 dBm to 0.0 dBm

Default Value: -50.0 dBm

Note

Reference level offset of the spectrum analyzer is not applied.

Return Value

Returns the status code of this operation.

The meaning of the status code is described in section Error (Status) Codes.

1.7.3.7.3 Configure Tracking Generator Frequency

C Function Prototype

```
ViStatus rssifs_confTrackingGeneratorFrequency (
      ViSession instrumentHandle,
      ViReal64 frequency);
```

Basic Function Prototype

```
Function rssifs_confTrackingGeneratorFrequency (
      ByVal instrumentHandle As ViSession,
      ByVal frequency As ViReal64) As ViStatus
```

Purpose

This function sets the discrete frequency at which is tracking generator tuned, when tracking generator state is Discrete, or frequency offset from spectrum analyzer frequency, when tracking generator state is Tracking.

Note

This function is available for FS315 only.

Parameters List

3. **ViSession instrumentHandle [in]**
   This control accepts the Instrument Handle returned by the Initialize function to select the desired instrument driver session.

   Default Value: None

4. **ViReal64 frequency [in]**
   Sets tracking generator frequency (frequency offset).

   Valid Range:
   FS315: 9.0e3 Hz to 3.0e9 Hz

   Default Value: 1.5e9 Hz

Note

Frequency offset of the spectrum analyzer is not applied.

Return Value

Returns the status code of this operation.

The meaning of the status code is described in section Error (Status) Codes.

1.7.3.7.4 Low-Level Functions

Description
Class of elementary functions to get (or set) tracking generator parameters value.

1.7.3.7.4.1 Get Tracking Generator State

C Function Prototype

```
ViStatus rssifs_getTrackingGeneratorState (
      ViSession instrumentHandle,
      ViInt32* trackingGeneratorState);
```

Basic Function Prototype

```
Function rssifs_getTrackingGeneratorState (
      ByVal instrumentHandle As ViSession,
      trackingGeneratorState As ViInt32) As ViStatus
```

Purpose
This function returns the state of tracking generator.

Note
This function is available for FS315 only.

Parameters List

3. **ViSession instrumentHandle [in]**
   This control accepts the Instrument Handle returned by the Initialize function to select the desired instrument driver session.

   Default Value: None

4. **ViInt32 trackingGeneratorState [out]**
   Returns tracking generator state.

   Valid Values:
   - 0 - Off
   - 1 - Tracking
   - 2 - Discrete

Return Value
Returns the status code of this operation.

The meaning of the status code is described in section Error (Status) Codes.

1.7.3.7.4.2 Get Tracking Generator Level

C Function Prototype

```
ViStatus rssifs_getTrackingGeneratorLevel (
      ViSession instrumentHandle,
      ViReal64* level);
```

Basic Function Prototype

```
Function rssifs_getTrackingGeneratorLevel (
      ByVal instrumentHandle As ViSession,
      level As ViReal64) As ViStatus
```

Purpose
This function returns the level of tracking generator.

Note
This function is available for FS315 only.

Parameters List

3. **ViSession instrumentHandle [in]**
   This control accepts the Instrument Handle returned by the Initialize function to select the desired instrument driver session.

   Default Value: None

4. **ViReal64 level [out]**
   Returns the level of the tracking generator in dBm.

Return Value

Returns the status code of this operation.

The meaning of the status code is described in section Error (Status) Codes.

1.7.3.7.4.3 Get Tracking Generator Frequency

C Function Prototype

```
ViStatus rssifs_getTrackingGeneratorFrequency (
      ViSession instrumentHandle,
      ViReal64* frequency);
```

Basic Function Prototype

```
Function rssifs_getTrackingGeneratorFrequency (
      ByVal instrumentHandle As ViSession,
      frequency As ViReal64) As ViStatus
```

Purpose

This function returns the discrete frequency at which is tracking generator tuned, when tracking generator state is Discrete, or frequency offset from spectrum analyzer frequency, when tracking generator state is Tracking.

Note

This function is available for FS315 only.

Parameters List

3. **ViSession instrumentHandle [in]**
   This control accepts the Instrument Handle returned by the Initialize function to select the desired instrument driver session.

   Default Value: None

4. **ViReal64 frequency [out]**
   Returns tracking generator frequency (frequency offset) in Hz.

Return Value

Returns the status code of this operation.

The meaning of the status code is described in section Error (Status) Codes.

1.7.3.8 Sweep Settings

Description

This class of functions is used to operate the analyzer's sweep parameters.

1.7.3.8.1 Configure Sweep

C Function Prototype

```
ViStatus rssifs_confSweep (
      ViSession instrumentHandle,
      ViInt32 sweep,
      ViInt32 sweepCount);
```

Basic Function Prototype

```
Function rssifs_confSweep (
      ByVal instrumentHandle As ViSession,
      ByVal sweep As ViInt32,
      ByVal sweepCount As ViInt32) As ViStatus
```

Purpose

This function configures the sweep mode (Continuous Sweep, Single Sweep) and sweep count parameter.

Parameters List

1. **ViSession instrumentHandle [in]**

   This control accepts the Instrument Handle returned by the Initialize function to select the desired instrument driver session.

   Default Value: None

2. **ViInt32 sweep [in]**

   This control selects the sweep parameter.

   Valid Range:

   - 0 - Continuous Sweep
   - 1 - Single Sweep
   - 2 - Sweep Count Value

   Default Value: 0

Note

- Continuous Sweep, Single Sweep:
  Determines if the trigger system is continuously initiated (Continuous Sweep) or waiting for a trigger (Single Sweep - the number of sweeps is automatically set to 1).
- Sweep Count Value:
  Number of sweeps to be performed after a single sweep has been started. If Trace Average, Max Hold or Min Hold is activated, this also determines the number of averaging or maximum search procedures.

3. **ViInt32 sweepCount [in]**

   Number of sweeps to be performed after a single sweep has been started. If zero has been entered, one sweep is performed.

   Valid Range: 0 to 32768

   Default Value: 0

Return Value

Returns the status code of this operation.

The meaning of the status code is described in section Error (Status) Codes.

1.7.3.8.2 Configure Sweep Time

C Function Prototype

```
ViStatus rssifs_confSweepTime (
      ViSession instrumentHandle,
      ViReal64 sweepTime);
```

Basic Function Prototype

```
Function rssifs_confSweepTime (
      ByVal instrumentHandle As ViSession,
      ByVal sweepTime As ViReal64) As ViStatus
```

Purpose

This function controls the setting of the analyzer's sweep time. Sweep time is calculated automatically by measurement module if zero value sweep time is passed.

Note

The value may be changed upon enforcement to reflect the device's timing capabilities. An error message will be sent to the error queue if this change occurs (if warning checking is enabled).

This function aborts and then restarts current measurement again.

Parameters List

1. **ViSession instrumentHandle [in]**

   This control accepts the Instrument Handle returned by the Initialize function to select the desired instrument driver session.

   Default Value: None

2. **ViReal64 sweepTime [in]**

   Sets the sweep time.

   - Valid Range (FS300):
     SPAN > 1kHz: 0.0, 0.1 s to 1000.0 s
     SPAN = 0 Hz: 100.0e-6 s to 20.0 s
   - Valid Range (FS315):
     SPAN > 1kHz: 0.0, 0.03 s to 1000.0 s
     SPAN = 0 Hz: 5.0e-6 s to 10.0 s
     TRACKING: 0.0, 0.2 s to 1000.0 s

   Default Value: 0.0 s

Note

Sweep time is calculated automatically by module if zero value sweep time is passed.

Return Value

Returns the status code of this operation.

The meaning of the status code is described in section Error (Status) Codes.

1.7.3.8.3 Configure Sweep Points

C Function Prototype

```
ViStatus rssifs_confSweepPoints (
      ViSession instrumentHandle,
      ViInt32 sweepPoints);
```

Basic Function Prototype

```
Function rssifs_confSweepPoints (
      ByVal instrumentHandle As ViSession,
       ByVal sweepPoints As ViInt32) As ViStatus
```

Purpose

This function defines the number of software pixels produced per a single sweep (trace data).

Note

This function aborts and then restarts current measurement again.

Parameters List

1. **ViSession instrumentHandle [in]**

   This control accepts the Instrument Handle returned by the Initialize function to select the desired instrument driver session.

   Default Value: None

2. **ViInt32 sweepPoints [in]**

   Number of software pixels produced per a single sweep.

   Valid Range:

   - FS300: 16 to 2048
   - FS315: 128 to 1024

   Default Value: 250

Return Value

Returns the status code of this operation.

The meaning of the status code is described in section Error (Status) Codes.

Low-Level Functions

Description

A class of elementary functions to get (or set) the analyzer's single parameter values.

1.7.3.8.3.1 Get Sweep Count

C Function Prototype

```
ViStatus rssifs_getSweepCount (
      ViSession instrumentHandle,
      ViInt32* sweepCount);
```

Basic Function Prototype

```
Function rssifs_getSweepCount (
      ByVal instrumentHandle As ViSession,
      sweepCount As ViInt32) As ViStatus
```

Purpose

This function returns the current number of started sweeps used for trace cache data evaluation.

Parameters List

1. **ViSession instrumentHandle [in]**

   This control accepts the Instrument Handle returned by the Initialize function to select the desired instrument driver session.

   Default Value: None

2. **ViInt32 sweepCount [out]**

   This control returns the number of started sweeps.

Return Value

Returns the status code of this operation.

The meaning of the status code is described in section Error (Status) Codes.

1.7.3.8.3.2 Get Sweep Time

C Function Prototype

```
ViStatus rssifs_getSweepTime (
      ViSession instrumentHandle,
      ViReal64* sweepTime);
```

Basic Function Prototype

```
Function rssifs_getSweepTime (
      ByVal instrumentHandle As ViSession,
      sweepTime As ViReal64) As ViStatus
```

Purpose

This function returns the sweep time.

Parameters List

1. **ViSession instrumentHandle [in]**

   This control accepts the Instrument Handle returned by the Initialize function to select the desired instrument driver session.

   Default Value: None

2. **ViReal64 sweepTime [out]**

   Returns the sweep time in seconds.

Return Value

Returns the status code of this operation.

The meaning of the status code is described in section Error (Status) Codes.

1.7.3.8.3.3 Get Sweep Mode

C Function Prototype

```
ViStatus rssifs_getSweepMode (
      ViSession instrumentHandle,
      ViInt32* sweepMode);
```

Basic Function Prototype

```
Function rssifs_getSweepMode (
      ByVal instrumentHandle As ViSession,
      sweepMode As ViInt32) As ViStatus
```

Purpose

This function returns the analyzer's sweep mode.

Parameters List

1. **ViSession instrumentHandle [in]**

   This control accepts the Instrument Handle returned by the Initialize function to select the desired instrument driver session.

   Default Value: None

2. **ViInt32 sweepMode [out]**

   Returns the analyzer's sweep mode.

   Where:

   - 0 - Continuous Sweep
   - 1 - Single Sweep

Return Value

Returns the status code of this operation.

The meaning of the status code is described in section Error (Status) Codes.

1.7.3.8.3.4 Get Sweep Points

C Function Prototype

```
ViStatus rssifs_getSweepPoints (
      ViSession instrumentHandle,
      ViInt32* sweepPoints);
```

Basic Function Prototype

```
Function rssifs_getSweepPoints (
      ByVal instrumentHandle As ViSession,
      sweepPoints As ViInt32) As ViStatus
```

Purpose

This function returns number of software pixels produced per a single sweep.

Parameters List

1. **ViSession instrumentHandle [in]**

   This control accepts the Instrument Handle returned by the Initialize function to select the desired instrument driver session.

   Default Value: None

2. **ViInt32 sweepPoints [out]**

   Returns number of software pixels produced per a single sweep.

Return Value

Returns the status code of this operation.

The meaning of the status code is described in section Error (Status) Codes.

1.7.3.9 Trigger Settings

Description

This class of functions is used to operate the analyzer's trigger parameters.

1.7.3.9.1 Configure Trigger

C Function Prototype

```
ViStatus rssifs_confTrg (
      ViSession instrumentHandle,
      ViInt32 trigger,
      ViReal64 triggerLevel,
      ViBoolean triggerSlope);
```

Basic Function Prototype

```
Function rssifs_confTrg (
      ByVal instrumentHandle As ViSession,
      ByVal trigger As ViInt32,
      ByVal triggerLevel As ViReal64,
      ByVal triggerSlope As ViBoolean) As ViStatus
```

Purpose

This function configures the analyzer's trigger conditions used to execute sweep.

Parameters List

1. **ViSession instrumentHandle [in]**

   This control accepts the Instrument Handle returned by the Initialize function to select the desired instrument driver session.

   Default Value: None

2. **ViInt32 trigger [in]**

   This control selects the trigger condition.

   Valid Range:

   - 0 - Free Run
   - 1 - Video
   - 2 - External
   - 3 - Line (RESERVED)

   Default Value: 1

Note

- Free Run:
  Automatic triggering the next measurement at the end of theprevious one.
- Video:
  The next measurement is triggered by the detection of a signal at the video filter output (Trigger Slope > Rising/Falling, Trigger Level). This setting is allowed only in zero span mode.
- External:
  The next measurement is triggered by the signal at the external trigger input (Trigger Slope > Rising/Falling).

3. **ViReal64 triggerLevel [in]**

   Trigger level value.

   Valid Range (offset = 0.0): -110.0 dBm to 36.0 dBm

   Default Value: -60.0 dBm

4. **ViBoolean triggerSlope [in]**

   This control selects the slope of the trigger signal when the trigger source is selected.

   Valid Range:

   - VI_FALSE (0) - Rising Edge
   - VI_TRUE (1) - Falling Edge

   Default Value: VI_FALSE (0)

Return Value

Returns the status code of this operation.

The meaning of the status code is described in section Error (Status) Codes.

1.7.3.9.2 Configure Trigger Delay

C Function Prototype

```
ViStatus rssifs_confTrgDelay (
      ViSession instrumentHandle,
      ViReal64 triggerDelay);
```

Basic Function Prototype

```
Function rssifs_confTrgDelay (
      ByVal instrumentHandle As ViSession,
      ByVal triggerDelay As ViReal64) As ViStatus
```

Purpose

This function defines the length of the trigger delay.

Parameters List

1. **ViSession instrumentHandle [in]**

   This control accepts the Instrument Handle returned by the Initialize function to select the desired instrument driver session.

   Default Value: None

2. **ViReal64 triggerDelay [in]**

   This control defines the length of the trigger delay.

   Valid Range:

   - Normal sweep:
   – 0.0 s to 100.0 ms

   - Zero Span Mode:
   – SweepTime to 100.0 ms (for SweepTime <= 100.0 ms)
   – 100.0 ms to 100.0 ms (for SweepTime > 100.0 ms)

   Default Value: 0.0 s

Return Value

Returns the status code of this operation.

The meaning of the status code is described in section Error (Status) Codes.

1.7.3.9.3 Low-Level Functions

Description

A class of elementary functions to get (or set) the analyzer's single parameter values.

1.7.3.9.3.1 Get Trigger Delay

C Function Prototype

```
ViStatus rssifs_getTriggerDelay (
      ViSession instrumentHandle,
      ViReal64* triggerDelay);
```

Basic Function Prototype

```
Function rssifs_getTriggerDelay (
      ByVal instrumentHandle As ViSession,
      triggerDelay As ViReal64) As ViStatus
```

Purpose

This function returns the trigger delay.

Parameters List

1. **ViSession instrumentHandle [in]**

   This control accepts the Instrument Handle returned by the Initialize function to select the desired instrument driver session.

   Default Value: None

2. **ViReal64 triggerDelay [out]**

   Returns the trigger delay in seconds.

Return Value

Returns the status code of this operation.

The meaning of the status code is described in section Error (Status) Codes.

1.7.3.9.3.2 Get Trigger Source

C Function Prototype

```
ViStatus rssifs_getTriggerSource (
      ViSession instrumentHandle,
      ViInt32* triggerSource);
```

Basic Function Prototype

```
Function rssifs_getTriggerSource (
      ByVal instrumentHandle As ViSession,
      triggerSource As ViInt32) As ViStatus
```

Purpose

This function returns the trigger source.

Parameters List

1. **ViSession instrumentHandle [in]**

   This control accepts the Instrument Handle returned by the Initialize function to select the desired instrument driver session.

   Default Value: None

2. **ViInt32 triggerSource [out]**

   This control returns the trigger source.

   Valid Values:

   - 0 - Free Run
   - 1 - Video
   - 2 - External
   - 3 - Line

Return Value

Returns the status code of this operation.

The meaning of the status code is described in section Error (Status) Codes.

1.7.3.9.3.3 Get Trigger Level

C Function Prototype

```
ViStatus rssifs_getTriggerLevel (
    ViSession instrumentHandle,
    ViReal64* triggerLevel);
```

Basic Function Prototype

```
Function rssifs_getTriggerLevel (
    ByVal instrumentHandle As ViSession,
    triggerLevel As ViReal64) As ViStatus
```

Purpose

This function returns the trigger level value.

Parameters List

1. **ViSession instrumentHandle [in]**

   This control accepts the Instrument Handle returned by the Initialize function to select the desired instrument driver session.

   Default Value: None

2. **ViReal64 triggerLevel [out]**

   Returns trigger level value in dBm.

Return Value

Returns the status code of this operation.

The meaning of the status code is described in section Error (Status) Codes.

1.7.3.9.3.4 Get Trigger Slope

C Function Prototype

```
ViStatus rssifs_getTriggerSlope (
    ViSession instrumentHandle,
    ViBoolean* triggerSlope);
```

Basic Function Prototype

```
Function rssifs_getTriggerSlope (
    ByVal instrumentHandle As ViSession,
    triggerSlope As ViBoolean) As ViStatus
```

Purpose

This function returns the trigger slope.

Parameters List

1. **ViSession instrumentHandle [in]**

   This control accepts the Instrument Handle returned by the Initialize function to select the desired instrument driver session.

   Default Value: None

2. **ViBoolean triggerSlope [out]**

   This control returns the trigger slope.

   Valid Values:

   - VI_FALSE (0) - Rising Edge
   - VI_TRUE (1) - Falling Edge

Return Value

Returns the status code of this operation.

The meaning of the status code is described in section Error (Status) Codes.

1.7.3.10 Bandwidth Settings

Description

This class of functions is used to operate the analyzer's resolution and video bandwidth parameters.

1.7.3.10.1 Configure Bandwidth

C Function Prototype

```
ViStatus rssifs_configureBandwidth (
      ViSession instrumentHandle,
      ViInt32 resolutionBandwidth,
      ViInt32 videoBandwidth);
```

Basic Function Prototype

```
Function rssifs_configureBandwidth (
      ByVal instrumentHandle As ViSession,
      ByVal resolutionBandwidth As ViInt32,
      ByVal videoBandwidth As ViInt32) As ViStatus
```

Purpose

This function sets the resolution and video bandwidth parameters. This function is capable to handle relations between RBW (resolution bandwidth) and VBW (video bandwidth).

Note

For every bandwidth of the resolution filter, the instrument supports only several bandwidths of the video filter. If combination of Video Bandwidth and Resolution Bandwidth is not allowed, this function returns settings conflict error.

Parameters List

1. **ViSession instrumentHandle [in]**

   This control accepts the Instrument Handle returned by the Initialize functionto select the desired instrument driver session.

   Default Value: None

2. **ViInt32 resolutionBandwidth [in]**

   Sets resolution bandwidth.

   Valid Range:

0	-	Auto
1	- 200	Hz
2	- 300	Hz
3	- 500	Hz
4	- 1	kHz
5	- 2	kHz
6	- 3	kHz
7	- 5	kHz
8	- 10	kHz
9	- 20	kHz
10	- 30	kHz
11	- 50	kHz
12	- 100	kHz
13	- 200	kHz
14	- 300	kHz
15	- 500	kHz
16	- 1	MHz
17	- 2	MHz (FS315 only)
18	- 3	MHz (FS315 only)
19	- 5	MHz (FS315 only)
20	- 10	MHz (FS315 only)
21	- 20	MHz (FS315 only)

Default Value: 0

3. **ViInt32 videoBandwidth [in]**

Sets the video bandwidth.

Valid Range:

-1	-	Video Filter Off (FS300 only)
0	-	Auto
1	- 10	Hz
2	- 20	Hz
3	- 30	Hz
4	- 50	Hz
5	- 100	Hz
6	- 200	Hz
7	- 300	Hz
8	- 500	Hz
9	- 1	kHz
10	- 2	kHz
11	- 3	kHz
12	- 5	kHz
13	- 10	kHz
14	- 20	kHz
15	- 30	kHz
16	- 50	kHz
17	- 100	kHz
18	- 200	kHz
19	- 300	kHz
20	- 500	kHz
21	- 1	MHz
22	- 2	MHz (FS315 only)
23	- 3	MHz (FS315 only)
24	- 5	MHz (FS315 only)
25	- 10	MHz (FS315 only)
26	- 20	MHz (FS315 only)

Default Value: 0

Return Value

Returns the status code of this operation.

The meaning of the status code is described in section Error (Status) Codes.

1.7.3.10.2 Configure Resolution Bandwidth

C Function Prototype

```
ViStatus rssifs_confResBW (
      ViSession instrumentHandle,
      ViReal64 resolutionBandwidth);
```

Basic Function Prototype

```
Function rssifs_confResBW (
      ByVal instrumentHandle As ViSession,
      ByVal resolutionBandwidth As ViReal64) As ViStatus
```

Purpose

This function manually controls the setting of the analyzer's resolution filter bandwidths.

Note

For every bandwidth of resolution filter, the instrument supports only several bandwidths of the video filter. If the combination of Video Bandwidth and Resolution Bandwidth is not allowed, this function returns settings conflict error.

This function sets resolution bandwidth to manual mode.

Parameters List

1. **ViSession instrumentHandle [in]**

   This control accepts the Instrument Handle returned by the Initialize function to select the desired instrument driver session.

   Default Value: None

2. **ViReal64 resolutionBandwidth [in]**

   Sets manually the resolution bandwidth.

   - FS300 supported resolution bandwidths in Hz:

   2.0e2, 3.0e2, 5.0e2, 1.0e3, 2.0e3, 3.0e3, 5.0e3, 1.0e4, 2.0e4, 3.0e4, 5.0e4, 1.0e5, 2.0e5, 3.0e5, 5.0e5, 1.0e6

   - FS315 supported resolution bandwidths in Hz:

   2.0e2, 3.0e2, 5.0e2, 1.0e3, 2.0e3, 3.0e3, 5.0e3, 1.0e4, 2.0e4, 3.0e4, 5.0e4, 1.0e5, 2.0e5, 3.0e5, 5.0e5, 1.0e6, 2.0e6, 3.0e6, 5.0e6, 1.0e7, 2.0e7

   Default Value: 1.0e6 Hz

Note

Passed value is coerced to the nearest acceptable value listed above.

Return Value

Returns the status code of this operation.

The meaning of the status code is described in section Error (Status) Codes.

1.7.3.10.3 Configure Video Bandwidth

C Function Prototype

```
ViStatus rssifs_confVideoBW (
      ViSession instrumentHandle,
      ViReal64 videoBandwidth);
```

Basic Function Prototype

```
Function rssifs_confVideoBW (
      ByVal instrumentHandle As ViSession,
      ByVal videoBandwidth As ViReal64) As ViStatus
```

Purpose

This function manually controls the setting of the analyzer's video filter bandwidths.

Note

For every bandwidth of resolution filter, the instrument supports only several bandwidths of the video filter. If the combination of Video Bandwidth and Resolution Bandwidth is not allowed, this function returns settings conflict error.

This function sets video bandwidth to manual mode.

Parameters List

1. **ViSession instrumentHandle [in]**

   This control accepts the Instrument Handle returned by the Initialize function to select the desired instrument driver session.

   Default Value: None

2. **ViReal64 videoBandwidth [in]**

   Sets manually the video bandwidth. Pass -1 to switch Video Filter off.

   - FS300 supported video bandwidths in Hz:

   1.0e1, 2.0e1, 3.0e1, 5.0e1, 1.0e2, 2.0e2, 3.0e2, 5.0e2, 1.0e3, 2.0e3, 3.0e3, 5.0e3, 1.0e4, 2.0e4, 3.0e4, 5.0e4, 1.0e5, 2.0e5, 3.0e5, 5.0e5, 1.0e6

   - FS315 supported video bandwidths in Hz:

   1.0e1, 2.0e1, 3.0e1, 5.0e1, 1.0e2, 2.0e2, 3.0e2, 5.0e2, 1.0e3, 2.0e3, 3.0e3, 5.0e3, 1.0e4, 2.0e4, 3.0e4, 5.0e4, 1.0e5, 2.0e5, 3.0e5, 5.0e5, 1.0e6, 2.0e6, 3.0e6, 5.0e6, 1.0e7, 2.0e7

   Default Value: 1.0e6 Hz

Note

Passed value is coerced to the nearest acceptable value listed above.

For FS300 pass -1 to switch Video Filter off.

Return Value

Returns the status code of this operation.

The meaning of the status code is described in section Error (Status) Codes.

1.7.3.10.4 Configure RBW vs Span Coupling

C Function Prototype

```
ViStatus rssifs_configureRBWSpanCoupling (
      ViSession instrumentHandle,
      ViInt32 rbwvsSpan);
```

Basic Function Prototype

```
Function rssifs_configureRBWSpanCoupling (
      ByVal instrumentHandle As ViSession,
      ByVal rbwvsSpan As ViInt32) As ViStatus
```

Purpose

This function changes the automatic coupling between the SPAN and resolution bandwidth (RBW). It is possible to switch the analyzer between the two settings "Normal" and "Low Noise" for even more accurate signal analysis, for example.

Note

The setting becomes effective only if the resolution bandwidth (RBW) is in AUTO mode.

If Low Noise mode is selected, high sensitivity of RF input is automatically set whenever RBW or Span parameter is changed.

Parameters List

1. **ViSession instrumentHandle [in]**

   This control accepts the Instrument Handle returned by the Initialize function to select the desired instrument driver session.

   Default Value: None

2. **ViInt32 rbwvsSpan [in]**

   Changes the automatic coupling between the SPAN and resolution bandwidth (RBW).

   Valid Range:

   - RSSIFS_RBW_SPAN_COUPLING_NORMAL (0) - Normal
   - RSSIFS_RBW_SPAN_COUPLING_LOW_NOISE (1) - Low Noise

   Default Value: RSSIFS_RBW_SPAN_COUPLING_NORMAL (0)

Note

- Normal:
  Corresponds to the normal operating mode. If RBW is in AUTO mode then settings is affected using following formula:

  RBW = SPAN / 50

  where result is coerced to the nearest acceptable value.

- Low Noise:
  If the span is 1 GHz or lower, the resolution bandwidth is decreased in the "Low Noise" setting as compared with the "Normal" setting. As a result, the sweep time increases simultaneously. The resolution bandwidths (RBWs) are set in accordance with the table below (SPAN ... RBW):

	SPAN >	1 GHz ...	1 MHz
1 GHz >=	SPAN >	50 MHz ...	300 kHz
50 MHz >=	SPAN >	10 MHz ...	100 kHz
10 MHz >=	SPAN >	5 MHz ...	30 kHz
5 MHz >=	SPAN >	1 MHz ...	10 kHz
1 MHz >=	SPAN >	200 kHz ...	3 kHz
200 kHz >=	SPAN >	100 kHz ...	1 kHz
100 kHz >=	SPAN >	50 kHz ...	500 Hz
50 kHz >=	SPAN >	20 kHz ...	300 Hz
20 kHz >=	SPAN >	1 kHz ...	200 Hz

Note Note that high sensitivity of RF input is selected (Input Attenuation and Reference Level parameters are changed).

Return Value Returns the status code of this operation.

The meaning of the status code is described in section Error (Status) Codes.

1.7.3.10.5 Configure RBW vs VBW Coupling

C Function Prototype

```
ViStatus rssifs_configureRBWVBWCoupling (
      ViSession instrumentHandle,
      ViReal64 couplingRatio);
```

Basic Function Prototype

```
Function rssifs_configureRBWVBWCoupling (
      ByVal instrumentHandle As ViSession,
      ByVal couplingRatio As ViReal64) As ViStatus
```

Purpose This function changes the automatic coupling between the resolution bandwidth (RBW) and video bandwidth (VBW).

The following coupling ratios are often available:

Sine signals ... RBW/VBW = 0.3 to 1
Pulse signals ... RBW/VBW = 0.1
Noise signals ... RBW/VBW = 10

Note The setting becomes effective only if the resolution bandwidth (RBW) or video bandwidth (VBW) is in AUTO mode.

Parameters List

1. **ViSession instrumentHandle [in]**
   This control accepts the Instrument Handle returned by the Initialize function to select the desired instrument driver session.

   Default Value: None

2. **ViReal64 couplingRatio [in]**
   Sets coupling ratio between resolution bandwidth (RBW) and video bandwidth (VBW) (RBW/VBW).

   Valid Range: 0.01 to 1000.0

   Default Value: 0.33

Note To ensure steady state of the video filter despite the reduced sweep time, the video bandwidth selected should be about three times greater than the resolution bandwidth (RBW/VBW = 0.3).

Return Value Returns the status code of this operation.

The meaning of the status code is described in section Error (Status) Codes.

1.7.3.10.6 Low-Level Functions

Description

A class of elementary functions to get (or set) the analyzer's single parameter values.

1.7.3.10.6.1 Get Resolution Bandwidth

C Function Prototype

```
ViStatus rssifs_getResolutionBandwidth (
      ViSession instrumentHandle,
      ViReal64* resolutionBandwidth);
```

Basic Function Prototype

```
Function rssifs_getResolutionBandwidth (
      ByVal instrumentHandle As ViSession,
      resolutionBandwidth As ViReal64) As ViStatus
```

Purpose

This function returns the resolution bandwidth.

Parameters List

1. **ViSession instrumentHandle [in]**

   This control accepts the Instrument Handle returned by the Initialize function to select the desired instrument driver session.

   Default Value: None

2. **ViReal64 resolutionBandwidth [out]**

   Returns resolution bandwidth in Hz.

Return Value

Returns the status code of this operation.

The meaning of the status code is described in section Error (Status) Codes.

1.7.3.10.6.2 Get Video Bandwidth

C Function Prototype

```
ViStatus rssifs_getVideoBandwidth (
      ViSession instrumentHandle,
      ViReal64* videoBandwidth);
```

Basic Function Prototype

```
Function rssifs_getVideoBandwidth (
      ByVal instrumentHandle As ViSession,
      videoBandwidth As ViReal64) As ViStatus
```

Purpose

This function returns the video bandwidth.

Parameters List

1. **ViSession instrumentHandle [in]**

   This control accepts the Instrument Handle returned by the Initialize function to select the desired instrument driver session.

   Default Value: None

2. **ViReal64 videoBandwidth [out]**

   Returns the video bandwidth in Hz. Returns -1 if Video Filter is switched off.

Return Value

Returns the status code of this operation.

The meaning of the status code is described in section Error (Status) Codes.

1.7.3.10.6.3 Get RBW vs Span Coupling Mode

C Function Prototype

```
ViStatus rssifs_getRBWSpanCouplingMode (
      ViSession instrumentHandle,
      ViInt32* rbwvsSpan);
```

Basic Function Prototype

```
Function rssifs_getRBWSpanCouplingMode (
      ByVal instrumentHandle As ViSession,
      rbwvsSpan As ViInt32) As ViStatus
```

Purpose

This function returns if the automatic coupling between the SPAN and resolution bandwidth (RBW) is set to "Normal" or "Low Noise" mode.

Parameters List

1. **ViSession instrumentHandle [in]**

   This control accepts the Instrument Handle returned by the Initialize function to select the desired instrument driver session.

   Default Value: None

2. **ViInt32 rbwvsSpan [out]**

   Returns if the automatic coupling between the SPAN and resolution bandwidth (RBW) is set to "Normal" or "Low Noise" mode.

   Valid Range:

   - RSSIFS_RBW_SPAN_COUPLING_NORMAL (0) - Normal
   - RSSIFS_RBW_SPAN_COUPLING_LOW_NOISE (1) - Low Noise

Note

- Normal:
  Corresponds to the normal operating mode. If RBW is in AUTO mode then settings is affected using following formula:

  RBW = SPAN / 50

  where result is coerced to the nearest acceptable value.

- Low Noise:
  If the span is 1 GHz or lower, the resolution bandwidth is decreased in the "Low Noise" setting as compared with the "Normal" setting. As a result, the sweep time increases simultaneously. The resolution bandwidths (RBWs) are set in accordance with the table below (SPAN ... RBW):

	SPAN >	1 GHz ...	1 MHz
1 GHz >=	SPAN >	50 MHz ...	300 kHz
50 MHz >=	SPAN >	10 MHz ...	100 kHz
10 MHz >=	SPAN >	5 MHz ...	30 kHz
5 MHz >=	SPAN >	1 MHz ...	10 kHz
1 MHz >=	SPAN >	200 kHz ...	3 kHz
200 kHz >=	SPAN >	100 kHz ...	1 kHz
100 kHz >=	SPAN >	50 kHz ...	500 Hz
50 kHz >=	SPAN >	20 kHz ...	300 Hz
20 kHz >=	SPAN >	1 kHz ...	200 Hz

Note

Note that high sensitivity of RF input is selected (Input Attenuation and Reference Level parameters are changed).

Return Value

Returns the status code of this operation.

The meaning of the status code is described in section Error (Status) Codes.

1.7.3.10.6.4 Get RBW vs VBW Coupling

C Function Prototype

```
ViStatus rssifs_getRBWVBWCoupling (
      ViSession instrumentHandle,
      ViReal64* couplingRatio);
```

Basic Function Prototype

```
Function rssifs_getRBWVBWCoupling (
      ByVal instrumentHandle As ViSession,
      couplingRatio As ViReal64) As ViStatus
```

Purpose

This function returns the automatic coupling between the resolution bandwidth (RBW) and video bandwidth (VBW).

Parameters List

3. **ViSession instrumentHandle [in]**
   This control accepts the Instrument Handle returned by the Initialize function to select the desired instrument driver session.

   Default Value: None

4. **ViReal64 couplingRatio [out]**
   Returns coupling ratio between resolution bandwidth (RBW) and video bandwidth (VBW) (RBW/VBW).

Return Value

Returns the status code of this operation.

The meaning of the status code is described in section Error (Status) Codes.

1.7.3.11 System Settings

Description

This class of functions is used to operate the analyzer's system settings.

1.7.3.11.1 Configure Reference Oscillator Source

C Function Prototype

```
ViStatus rssifs_confReferenceOsc (
      ViSession instrumentHandle,
      ViBoolean referenceOscillatorSource);
```

Basic Function Prototype

```
Function rssifs_confReferenceOsc (
      ByVal instrumentHandle As ViSession,
      ByVal referenceOscillatorSource As ViBoolean) As ViStatus
```

Purpose

This function selects the reference oscillator source (internal TCXO/OCXO or external).

Parameters List

1. **ViSession instrumentHandle [in]**

   This control accepts the Instrument Handle returned by the Initialize function to select the desired instrument driver session.

   Default Value: None

2. **ViBoolean referenceOscillatorSource [in]**

   Selects source of the reference oscillator.

   Valid Range:

   - VI_FALSE (0) - Internal (TCXO/OCXO)
   - VI_TRUE (1) - External

   Default Value: VI_FALSE (0)

Return Value

Returns the status code of this operation.

The meaning of the status code is described in section Error (Status) Codes.

1.7.3.11.2 Configure Transducer Factor

C Function Prototype

```
ViStatus rssifs_confTransducerFactor (
      ViSession instrumentHandle,
      ViBoolean transducerFactors);
```

Basic Function Prototype

```
Function rssifs_confTransducerFactor (
      ByVal instrumentHandle As ViSession,
      ByVal transducerFactors As ViBoolean) As ViStatus
```

Purpose

This function activates or deactivated usage of transducer factors.

Parameters List

3. **ViSession instrumentHandle [in]**
   This control accepts the Instrument Handle returned by the Initialize function to select the desired instrument driver session.

   Default Value: None

4. **ViBoolean transducerFactors [in]**
   Activates or deactivated usage of transducer factors.

   Valid Range:
   - VI_FALSE (0) - Off (Default Value)
   - VI_TRUE (1) - On

Return Value

Returns the status code of this operation.

The meaning of the status code is described in section Error (Status) Codes.

1.7.3.11.3 Configure Transducer Factor Values

C Function Prototype

```
ViStatus rssifs_confTransducerFactorValues (
      ViSession instrumentHandle,
      ViInt32 noofValues,
      ViReal64[] frequencyValues,
      ViReal64[] levelValues,
      ViInt32 unit);
```

Basic Function Prototype

```
Function rssifs_confTransducerFactorValues (
      ByVal instrumentHandle As ViSession,
      ByVal noofValues As ViInt32,
      frequencyValues As ViReal64,
      levelValues As ViReal64,
      ByVal unit As ViInt32) As ViStatus
```

Purpose

This function is used to define transducer factor values.

Note

- Transducer factors for a sweep are calculated once in advance for every point displayed and are added to the result of the level measurement during the sweep. If the sweep range changes, the correction values are calculated again.
- If the transducer factor is not defined for the entire sweep range, the values missing are replaced by zeroes.
- This function applies over the trace cache data.

Parameters List

6. **ViSession instrumentHandle [in]**
   This control accepts the Instrument Handle returned by the Initialize function to select the desired instrument driver session.

Default Value: None

7. **ViInt32 noofValues [in]**
   Specifies the number of frequency and level pairs (corresponds with number of array items).

   Valid Range: 1 to 1024

   Default Value: none

8. **ViReal64[] frequencyValues [in]**
   This array defines the frequency part (in Hz) of the test points for the transducer factor. The frequencies must be in ascending order.

   Valid Range: 0.0 Hz to 200.0 GHz

   Default Value: none

   Note:

   The frequencies entered may exceed the frequency range of the instrument since only the set frequency range is taken into account for measurements.

9. **ViReal64[] levelValues [in]**
   This array defines the level part of the test points for the transducer factor. The level values are sent as dimensionless numbers; the unit is specified by control Unit.

   Valid Range: not checked

   Default Value: none

   Note:

   Gain has to be entered as a negative value, and attenuation as a positive value.

10. **ViInt32 unit [in]**
    Defines the transducer unit.

    Valid Range:
    - RSSIFS_TRD_UNIT_DB (0) - dB
    - RSSIFS_TRD_UNIT_DBUV (1) - dBuV
    - RSSIFS_TRD_UNIT_DBUA (2) - dBuA
    - RSSIFS_TRD_UNIT_DBPT (3) - dBpT
    - RSSIFS_TRD_UNIT_DBUVM (4) - dBuV/m
    - RSSIFS_TRD_UNIT_DBUAM (5) - dBuA/m

    Default Value: RSSIFS_TRD_UNIT_DB (0)

Return Value

Returns the status code of this operation.

The meaning of the status code is described in section Error (Status) Codes.

1.7.3.11.4 Low-Level Functions

Description Class of elementary functions to get (or set) analyzer's parameter value.

1.7.3.11.4.1 Get Transducer Factor

C Function Prototype

```
ViStatus rssifs_getTransducerFactor (
      ViSession instrumentHandle,
      ViBoolean* transducerFactors);
```

Basic Function Prototype

```
Function rssifs_getTransducerFactor (
      ByVal instrumentHandle As ViSession,
      transducerFactors As ViBoolean) As ViStatus
```

Purpose This function returns whether usage of transducer factors is activated or deactivated.

Parameters List

3. **ViSession instrumentHandle [in]**
   This control accepts the Instrument Handle returned by the Initialize function to select the desired instrument driver session.

   Default Value: None

4. **ViBoolean transducerFactors [out]**
   Returns whether usage of transducer factors is activated or deactivated.

   Valid Range:
   - VI_FALSE (0) - Off
   - VI_TRUE (1) - On

Return Value Returns the status code of this operation.

The meaning of the status code is described in section Error (Status) Codes.

1.7.3.11.4.2 Get Transducer Factor Values

C Function Prototype

```
ViStatus rssifs_getTransducerFactorValues (
      ViSession instrumentHandle,
      ViInt32 noofValues,
      ViReal64[] frequencyValues,
      ViReal64[] levelValues,
      ViInt32* unit);
```

Basic Function Prototype

```
Function rssifs_getTransducerFactorValues (
      ByVal instrumentHandle As ViSession,
      ByVal noofValues As ViInt32,
      frequencyValues As ViReal64,
      levelValues As ViReal64,
      unit As ViInt32) As ViStatus
```

Purpose This function returns currently defined transducer factor values.

Parameters List

6. **ViSession instrumentHandle [in]**
   This control accepts the Instrument Handle returned by the Initialize function to select the desired instrument driver session.

   Default Value: None

7. **ViInt32 noofValues [in]**
   Specifies the number of frequency and level pairs (corresponds with number of array items).

   Valid Range: 1 to 1024

Default Value: none

8. ViReal64[] frequencyValues [out]

Returns the frequency part (in Hz) of the test points of the transducer factor.

Note:

The array must contain at least number of elements defined with parameter 'No of Values'.

9. ViReal64[] levelValues [out]

Returns the level part of the test points of the transducer factor. The level values are sent as dimensionless numbers; the unit is specified by indicator Unit.

Note:

The array must contain at least number of elements defined with parameter 'No of Values'.

10. ViInt32 unit [out]

Returns the transducer unit.

Valid Range:

- RSSIFS_TRD_UNIT_DB (0) - dB
- RSSIFS_TRD_UNIT_DBUV (1) - dBuV
- RSSIFS_TRD_UNIT_DBUA (2) - dBuA
- RSSIFS_TRD_UNIT_DBPT (3) - dBpT
- RSSIFS_TRD_UNIT_DBUVM (4) - dBuV/m
- RSSIFS_TRD_UNIT_DBUAM (5) - dBuA/m

Return Value

Returns the status code of this operation.

The meaning of the status code is described in section Error (Status) Codes.

1.7.3.12 Measurement Functions

Description

This class of functions is used to operate analyzer's measurement functions parameters.

1.7.3.12.1 Configure Channel Power Measurement

C Function Prototype

```
ViStatus rssifs_confChannelPowerMeasurement (
      ViSession instrumentHandle,
      ViBoolean measurementState,
      ViReal64 channelBandwidth);
```

Basic Function Prototype

```
Function rssifs_confChannelPowerMeasurement (
      ByVal instrumentHandle As ViSession,
      ByVal measurementState As ViBoolean,
      ByVal channelBandwidth As ViReal64) As ViStatus
```

Purpose

This function configures channel power measurement parameters.

Note

This function is only available in the frequency domain (span > 0).

Parameters List

4. **ViSession instrumentHandle [in]**

   This control accepts the Instrument Handle returned by the Initialize function to select the desired instrument driver session.

   Default Value: None

5. **ViBoolean measurementState [in]**

   Enables or disables channel power measurement.

   Valid Range:
   - VI_FALSE (0) - Off (Default Value)
   - VI_TRUE (1) - On

6. **ViReal64 channelBandwidth [in]**

   Sets channel bandwidth around the center frequency within which the channel power is computed.

   Valid Range: 1000.0 Hz to 3.0e9 Hz

   Default Value: 150.0e6 Hz

Return Value

Returns the status code of this operation.

The meaning of the status code is described in section Error (Status) Codes.

1.7.3.12.2 Configure Occupied Bandwidth Measurement

C Function Prototype

```
ViStatus rssifs_confOccupiedBandwidthMeasurement (
      ViSession instrumentHandle,
      ViBoolean measurementState,
      ViReal64 powerBandwidth);
```

Basic Function Prototype

```
Function rssifs_confOccupiedBandwidthMeasurement (
      ByVal instrumentHandle As ViSession,
      ByVal measurementState As ViBoolean,
      ByVal powerBandwidth As ViReal64) As ViStatus
```

Purpose

This function configures occupied bandwidth measurement parameters. The occupied bandwidth is defined as the bandwidth containing a defined percentage of the total transmitted power.

Note

This function is only available in the frequency domain (span > 0).

Parameters List

4. **ViSession instrumentHandle [in]**
   This control accepts the Instrument Handle returned by the Initialize function to select the desired instrument driver session.

   Default Value: None

5. **ViBoolean measurementState [in]**
   Enables or disables occupied bandwidth measurement.

   Valid Range:
   - VI_FALSE (0) - Off (Default Value)
   - VI_TRUE (1) - On

6. **ViReal64 powerBandwidth [in]**
   Defines the percentage of power related to the total power in the measured frequency range which defines the occupied bandwidth (percentage of total power).

   Valid Range: 10.0 % to 99.9 %

   Default Value: 99.0 %

Return Value

Returns the status code of this operation.

The meaning of the status code is described in section Error (Status) Codes.

1.7.3.12.3 Configure Time Domain Power Measurement

C Function Prototype

```
ViStatus rssifs_confTimeDomainPowerMeasurement (
      ViSession instrumentHandle,
      ViBoolean measurementState,
      ViInt32 powerMeasurement);
```

Basic Function Prototype

```
Function rssifs_confTimeDomainPowerMeasurement (
      ByVal instrumentHandle As ViSession,
      ByVal measurementState As ViBoolean,
      ByVal powerMeasurement As ViInt32) As ViStatus
```

Purpose

This function configures time domain power measurement parameters to determine the power of the signal in the time domain (SPAN = 0 Hz) by summing up the power at the individual pixels and dividing the result by the number of pixels.

Note

- Measurement limits (configured limit lines) might apply.
- This function is only available in the time domain (zero span mode).

Parameters List

4. **ViSession instrumentHandle [in]**
   This control accepts the Instrument Handle returned by the Initialize function to select the desired instrument driver session.

   Default Value: None

5. **ViBoolean measurementState [in]**
   Enables or disables time domain power measurement.

   Valid Range:
   - VI_FALSE (0) - Off (Default Value)
   - VI_TRUE (1) - On

6. **ViInt32 powerMeasurement [in]**
   Selects whether the mean power or the rms power can be measured by means of the individual power values.

   Valid Values:
   - RSSIFS_TDOM_POWER_MEAN (0) - Mean
   - RSSIFS_TDOM_POWER_RMS (1) - RMS

   Default Value: RSSIFS_TDOM_POWER_MEAN (0)

Return Value

Returns the status code of this operation.

The meaning of the status code is described in section Error (Status) Codes.

1.7.3.12.4 Configure Limit Lines

C Function Prototype

```
ViStatus rssifs_confLimitLines (
      ViSession instrumentHandle,
      ViInt32 lineType,
      ViInt32 lineNumber,
      ViBoolean lineState,
      ViReal64 linePositionValue);
```

Basic Function Prototype

```
Function rssifs_confLimitLines (
      ByVal instrumentHandle As ViSession,
      ByVal lineType As ViInt32,
      ByVal lineNumber As ViInt32,
      ByVal lineState As ViBoolean,
      ByVal linePositionValue As ViReal64) As ViStatus
```

Purpose

This function sets the limit lines used to define measuring interval.

Parameters List

6. **ViSession instrumentHandle [in]**
   This control accepts the Instrument Handle returned by the Initialize function to select the desired instrument driver session.

   Default Value: None

7. **ViInt32 lineType [in]**
   Selects limit line type.

   Valid Range:
   - 0 - Time Line
   - 1 - Frequency Line

Note

- Currently two vertical frequency (time) lines for indicating frequencies (times) or for determining frequency or time operation ranges.
- The frequency lines are only valid for a SPAN > 0.
- The time lines are only valid for a SPAN = 0.

Parameters List

8. ViInt32 lineNumber [in]

This control defines the limit line number.

Valid Values: 1, 2

Default Value: 1

9. ViBoolean lineState [in]

Enables or disables limit lines.

Valid Range:

- VI_FALSE (0) - Off (Default Value)
- VI_TRUE (1) - On

10. ViReal64 linePositionValue [in]

This control sets the limit line position value. This value depends on limit line type.

Valid Range:

- Frequency Line (offset = 0.0): 0.0 Hz to 3.0e9 Hz
- Time Line: 0.0 to 1000.0 s

Default Value: 0.0

Note

Value of limit line 1 should be less or equal to the value of limit line 2.

Return Value

Returns the status code of this operation.

The meaning of the status code is described in section Error (Status) Codes.

1.7.3.12.5 Low-Level Functions

Description

Class of elementary functions to get (or set) analyzer's parameter value.

1.7.3.12.5.1 Get Channel Power Measurement

C Function Prototype

```
ViStatus rssifs_getChannelPowerMeasurement (
    ViSession instrumentHandle,
    ViBoolean* measurementState,
    ViReal64* channelBandwidth);
```

Basic Function Prototype

```
Function rssifs_getChannelPowerMeasurement (
    ByVal instrumentHandle As ViSession,
    measurementState As ViBoolean,
    channelBandwidth As ViReal64) As ViStatus
```

Purpose

This function returns channel power measurement parameters.

Note

This function is only available in the frequency domain (span > 0).

Parameters List

4. ViSession instrumentHandle [in]

This control accepts the Instrument Handle returned by the Initialize function to select the desired instrument driver session.

Default Value: None

5. ViBoolean measurementState [out]

Returns whether channel power measurement is enabled or disabled.

Valid Range:

- VI_FALSE (0) - Off
- VI_TRUE (1) - On

6. ViReal64 channelBandwidth [out]

Returns channel bandwidth around the center frequency within which the channel power is computed (in Hz).

Return Value

Returns the status code of this operation.

The meaning of the status code is described in section Error (Status) Codes.

1.7.3.12.5.2 Get Occupied Bandwidth Measurement

C Function Prototype

```
ViStatus rssifs_getOccupiedBandwidthMeasurement (
      ViSession instrumentHandle,
      ViBoolean* measurementState,
      ViReal64* powerBandwidth);
```

Basic Function Prototype

```
Function rssifs_getOccupiedBandwidthMeasurement (
      ByVal instrumentHandle As ViSession,
      measurementState As ViBoolean,
      powerBandwidth As ViReal64) As ViStatus
```

Purpose

This function returns occupied bandwidth measurement parameters.

Note

This function is only available in the frequency domain (span > 0).

Parameters List

4. **ViSession instrumentHandle [in]**
   This control accepts the Instrument Handle returned by the Initialize function to select the desired instrument driver session.

   Default Value: None

5. **ViBoolean measurementState [out]**
   Returns whether occupied bandwidth measurement is enabled or disabled.

   Valid Range:
   - VI_FALSE (0) - Off
   - VI_TRUE (1) - On

6. **ViReal64 powerBandwidth [out]**
   Returns the percentage of power related to the total power in the measured frequency range which defines the occupied bandwidth (percentage of total power).

Return Value

Returns the status code of this operation.

The meaning of the status code is described in section Error (Status) Codes.

1.7.3.12.5.3 Get Time Domain Power Measurement

C Function Prototype

```
ViStatus rssifs_getTimeDomainPowerMeasurement (
      ViSession instrumentHandle,
      ViBoolean* measurementState,
      ViInt32* powerMeasurement);
```

Basic Function Prototype

```
Function rssifs_getTimeDomainPowerMeasurement (
      ByVal instrumentHandle As ViSession,
      measurementState As ViBoolean,
      powerMeasurement As ViInt32) As ViStatus
```

Purpose

This function returns time domain power measurement parameters which determines the power of the signal in the time domain (SPAN = 0 Hz) by summing up the power at the individual pixels and dividing the result by the number of pixels.

Note

This function is only available in the time domain (zero span mode).

Parameters List

4. **ViSession instrumentHandle [in]**
   This control accepts the Instrument Handle returned by the Initialize function to select the desired instrument driver session.

   Default Value: None

5. **ViBoolean measurementState [out]**
   Returns whether time domain power measurement is enabled or disabled.

   Valid Range:
   - VI_FALSE (0) - Off
   - VI_TRUE (1) - On

6. **ViInt32 powerMeasurement [out]**
   Returns whether the mean power or the rms power can be measured by means of the individual power values.

   Valid Values:
   - RSSIFS_TDOM_POWER_MEAN (0) - Mean
   - RSSIFS_TDOM_POWER_RMS (1) - RMS

Return Value

Returns the status code of this operation.

The meaning of the status code is described in section Error (Status) Codes.

1.7.3.12.5.4 Get Limit Lines

C Function Prototype

```
ViStatus rssifs_getLimitLines (
      ViSession instrumentHandle,
      ViInt32 lineType,
      ViInt32 lineNumber,
      ViBoolean* lineState,
      ViReal64* linePositionValue);
```

Basic Function Prototype

```
Function rssifs_getLimitLines (
      ByVal instrumentHandle As ViSession,
      ByVal lineType As ViInt32,
      ByVal lineNumber As ViInt32,
      lineState As ViBoolean,
      linePositionValue As ViReal64) As ViStatus
```

Purpose

This function returns settings of the limit lines used to define measuring interval.

Parameters List

6. **ViSession instrumentHandle [in]**
   This control accepts the Instrument Handle returned by the Initialize function to select the desired instrument driver session.

   Default Value: None

7. **ViInt32 lineType [in]**
   Selects limit line type.

   Valid Range:
   0 - Time Line
   1 - Frequency Line

Note

- Currently two vertical frequency (time) lines for indicating frequencies (times) or for determining frequency or time operation ranges.
- The frequency lines are only valid for a SPAN > 0.
- The time lines are only valid for a SPAN = 0.¨

Parameters List

8. **ViInt32 lineNumber [in]**
   This control defines the limit line number.

   Valid Values: 1, 2

   Default Value: 1

9. **ViBoolean lineState [out]**
   Returns whether the selected limit line is enabled or disabled.

   Valid Range:
   - VI_FALSE (0) - Off
   - VI_TRUE (1) - On

10. **ViReal64 linePositionValue [out]**
    This control returns the position of selected limit linelimit line position value. This value and unit depends on limit line type.

Return Value

Returns the status code of this operation.

The meaning of the status code is described in section Error (Status) Codes.

1.7.4 Action/Status Functions

Description

This class of functions begins or terminates an acquisition.

1.7.4.1 Trigger Group

Description

This class contains trigger related functions.

1.7.4.1.1 Send Trigger

C Function Prototype

```
ViStatus rssifs_actSendTrg (
    ViSession instrumentHandle);
```

Basic Function Prototype

```
Function rssifs_actSendTrg (
    ByVal instrumentHandle As ViSession) As ViStatus
```

Purpose

This function triggers all actions waiting for a trigger event.

Measurement is started under configured trigger and sweep conditions.

Note

This function invalidates content of trace cache data. To validate it call Read Complete Sweep Data (rssifs_readCompleteSweepData).

Parameters List

1. **ViSession instrumentHandle [in]**

   This control accepts the Instrument Handle returned by the Initialize function to select the desired instrument driver session.

   Default Value: None

Return Value

Returns the status code of this operation.

The meaning of the status code is described in section Error (Status) Codes.

1.7.4.1.2 Send Trigger and Wait for OPC

C Function Prototype

```
ViStatus rssifs_actSendTrgWopc (
    ViSession instrumentHandle,
    ViInt32 timeout);
```

Basic Function Prototype

```
Function rssifs_actSendTrgWopc (
    ByVal instrumentHandle As ViSession,
    ByVal timeout As ViInt32) As ViStatus
```

Purpose

This function triggers all actions waiting for a trigger event and waits for all pending operation completed (OPC) before returning the status code.

Note

Content of trace cache data will be after completion valid.

Parameters List

1. **ViSession instrumentHandle [in]**

   This control accepts the Instrument Handle returned by the Initialize function to select the desired instrument driver session.

   Default Value: None

2. **ViInt32 timeout [in]**

   Sets the timeout for the triggering routine to be finished. If the length of time required for triggering exceeds the timeout value, then the function will return with a timeout error.

   Valid Range: 0 ms to 600000 ms

   Default Value: 10000 ms

Return Value

Returns the status code of this operation.

The meaning of the status code is described in section Error (Status) Codes.

1.7.4.1.3 Abort

C Function Prototype

```
ViStatus rssifs_actAbort (
      ViSession instrumentHandle);
```

Basic Function Prototype

```
Function rssifs_actAbort (
      ByVal instrumentHandle As ViSession) As ViStatus
```

Purpose

This function aborts a current measurement and enters Idle state immediately.

Note

This function invalidates content of trace cache data. To validate it call Send Trigger (rssifs_actSendTrg) and Read Complete Sweep Data (rssifs_readCompleteSweepData) or Send Trigger and Wait for OPC (rssifs_actSendTrgWopc).

Parameters List

1. **ViSession instrumentHandle [in]**

   This control accepts the Instrument Handle returned by the Initialize function to select the desired instrument driver session.

   Default Value: None

Return Value

Returns the status code of this operation.

The meaning of the status code is described in section Error (Status) Codes.

1.7.4.2 Calibration Group

Description

This class contains the calibration routines.

1.7.4.2.1 Calibration

C Function Prototype

```
ViStatus rssifs_actCalibration (
      ViSession instrumentHandle,
      ViInt32 timeout);
```

Basic Function Prototype

```
Function rssifs_actCalibration (
      ByVal instrumentHandle As ViSession,
      ByVal timeout As ViInt32) As ViStatus
```

Purpose

Performs automatic DC offset calibration. Parameter should be zero. User should be advised of taking signal off from the input.

Note

If the calibration fails, this function returns an error code.

This function is available for FS300 only.

Parameters List

1. **ViSession instrumentHandle [in]**

   This control accepts the Instrument Handle returned by the Initialize function to select the desired instrument driver session.

   Default Value: None

2. **ViInt32 timeout [in]**

   Sets the timeout for the calibration routine to be finished. If the length of time required for calibration exceeds the timeout value, then the function will return with a timeout error and the instrument will continue with calibration.

   Valid Range: 0 ms to 600000 ms

   Default Value: 30000 ms

Return Value

Returns the status code of this operation.

The meaning of the status code is described in section Error (Status) Codes.

1.7.4.3 Device Status Group

Description

The status reporting system functions provides information on the present operating state of the instrument, e.g. that the instrument is presently ready for immediate operation, working, performing self-test, etc.

1.7.4.3.1 Get Device State

C Function Prototype

```
ViStatus rssifs_getDeviceState (
      ViSession instrumentHandle,
      ViInt32* deviceState);
```

Basic Function Prototype

```
Function rssifs_getDeviceState (
      ByVal instrumentHandle As ViSession,
      deviceState As ViInt32) As ViStatus
```

Purpose

This function presents logical state (present operating state) of the device, e.g. that the instrument is presently ready for immediate operation, performing measurement task, performing self-test, etc.

Parameters List

3. **ViSession instrumentHandle [in]**
   This control accepts the Instrument Handle returned by the Initialize function to select the desired instrument driver session.

   Default Value: None

4. **ViInt32 deviceState [out]**
   Returns logical state (present operating state) of the device.

FS300 Spectrum Analyzer		
Value	*Name*	*Description*
0x0	Idle	Device is in power-saving mode and ready for immediate operation.
0x1	Busy	Device is working.
0x80	Sleep	Device is in battery saving mode. It may take some time to wake it up.
0x81	Init	Device is entering Idle mode (waking up from Sleep, booting or performing self test).

FS315 Spectrum Analyzer		
Value	*Name*	*Description*
0x0	Idle	Device is in idle mode and ready for immediate operation.
0x10	Meas	Device performs measurement task.
0x64	Service	Device was triggered to Service mode by service command.
0xFF	Init	Device performs initialization. It is entering Idle mode.
0xFFFF	Sleep	Device is in power-saving mode.

Return Value

Returns the status code of this operation.

The meaning of the status code is described in section Error (Status) Codes.

1.7.5 Data Functions

Description

This class of functions transfers data to or from the instrument.

1.7.5.1 Read Marker Counter Value

C Function Prototype

```
ViStatus rssifs_readMarkerCounterValue (
      ViSession instrumentHandle,
      ViInt32 markerNumber,
       ViReal64* markerFrequency);
```

Basic Function Prototype

```
Function rssifs_readMarkerCounterValue (
      ByVal instrumentHandle As ViSession,
      ByVal markerNumber As ViInt32,
      markerFrequency As ViReal64) As ViStatus
```

Purpose

This function gets frequency counter value from the marker position.

Note

Frequency counter must be turned to On!

This function does not send any triggering request to the instrument. To successfully proceed, measurement must be initiated prior to call this function.

Parameters List

1. **ViSession instrumentHandle [in]**

   This control accepts the Instrument Handle returned by the Initialize function to select the desired instrument driver session.

   Default Value: None

2. **ViInt32 markerNumber [in]**

   Selects the marker.

   Valid Value: 1 to 1

   Default Value: 1

3. **ViReal64 markerFrequency [out]**

   This control returns measured value of the frequency counter in Hz.

Return Value

Returns the status code of this operation.

The meaning of the status code is described in section Error (Status) Codes.

1.7.5.2 Read Marker Value

C Function Prototype

```
ViStatus rssifs_readMarkerValue (
      ViSession instrumentHandle,
      ViInt32 markerNumber,
      ViReal64* markerValue);
```

Basic Function Prototype

```
Function rssifs_readMarkerValue (
      ByVal instrumentHandle As ViSession,
      ByVal markerNumber As ViInt32,
      markerValue As ViReal64) As ViStatus
```

Purpose

This function gets the marker level value from current marker position over the trace cache data.

Note

Corresponding marker must be enabled!

Parameters List

1. **ViSession instrumentHandle [in]**

   This control accepts the Instrument Handle returned by the Initialize function to select the desired instrument driver session.

   Default Value: None

2. **ViInt32 markerNumber [in]**

   Selects the marker.

   Valid Value: 1 to 2

   Default Value: 1

3. **ViReal64 markerValue [out]**

   This control returns marker level value from corresponding marker position.

Note

Transferred value represents signal level at the marker position.

Value is returned in current trace unit. See Configure Trace Unit (rssifs_confTraceUnit) function.

Return Value

Returns the status code of this operation.

The meaning of the status code is described in section Error (Status) Codes.

1.7.5.3 Read Delta Marker Value

C Function Prototype

```
ViStatus rssifs_readDeltaMarkerValue (
      ViSession instrumentHandle,
      ViInt32 deltaMarkerNumber,
      ViReal64* deltaMarkerValue);
```

Basic Function Prototype

```
Function rssifs_readDeltaMarkerValue (
      ByVal instrumentHandle As ViSession,
      ByVal deltaMarkerNumber As ViInt32,
      deltaMarkerValue As ViReal64) As ViStatus
```

Purpose

This function returns delta marker level value relative to the corresponding normal marker position over the trace cache data.

Note

Corresponding marker and delta marker must be enabled!

Parameters List

4. **ViSession instrumentHandle [in]**
   This control accepts the Instrument Handle returned by the Initialize function to select the desired instrument driver session.

   Default Value: None

5. **ViInt32 deltaMarkerNumber [in]**
   Selects delta marker.

   Valid Value: 1 to 2

   Default Value: 1

6. **ViReal64 deltaMarkerValue [out]**
   This control returns delta marker level value relative to the corresponding normal marker position.

Note

Value is returned in current trace unit. See Configure Trace Unit (rssifs_confTraceUnit) function.

Return Value

Returns the status code of this operation.

The meaning of the status code is described in section Error (Status) Codes.

1.7.5.4 Read N dB Down Marker Value

C Function Prototype

```
ViStatus rssifs_readNdBDownMarkerValue (
      ViSession instrumentHandle,
      ViInt32 markerNumber,
      ViReal64[] spacingValue);
```

Basic Function Prototype

```
Function rssifs_readNdBDownMarkerValue (
      ByVal instrumentHandle As ViSession,
      ByVal markerNumber As ViInt32,
      spacingValue As ViReal64) As ViStatus
```

Purpose

This function queries measurement result of the temporary markers which are n dB below the active reference marker.

Note

Corresponding marker must be enabled!

Parameters List

4. **ViSession instrumentHandle [in]**
   This control accepts the Instrument Handle returned by the Initialize function to select the desired instrument driver session.

   Default Value: None

5. **ViInt32 markerNumber [in]**
   Selects the marker.

   Valid Value: 1 to 2

   Default Value: 1

6. **ViReal64[] spacingValue [out]**
   This control returns frequency spacing (bandwidth in Hz) (or time distance if zero span in seconds) and the temporary markers positions which are n dB below the active reference marker.

   Where results are in order: <spacing>,<start_pos>,<stop_pos>

Note

The array must contain at least 3 elements.

Return Value

Returns the status code of this operation.

The meaning of the status code is described in section Error (Status) Codes.

1.7.5.5 Read Noise Marker Value

C Function Prototype

```
ViStatus rssifs_readNoiseMarkerValue (
      ViSession instrumentHandle,
      ViInt32 markerNumber,
      ViReal64* markerValue);
```

Basic Function Prototype

```
Function rssifs_readNoiseMarkerValue (
      ByVal instrumentHandle As ViSession,
      ByVal markerNumber As ViInt32,
      markerValue As ViReal64) As ViStatus
```

Purpose

This function performs noise power density measurement at the current marker position over the trace cache data.

Note

Corresponding marker must be enabled!

Parameters List

4. **ViSession instrumentHandle [in]**
   This control accepts the Instrument Handle returned by the Initialize function to select the desired instrument driver session.

   Default Value: None

5. **ViInt32 markerNumber [in]**
   Selects the marker.

   Valid Value: 1 to 2

   Default Value: 1

6. **ViReal64 markerValue [out]**
   Returns noise power density measurement result (dBm/Hz) at the current marker position over the trace cache data.

Note

Return Value

Returns the status code of this operation.

The meaning of the status code is described in section Error (Status) Codes.

1.7.5.6 Read Channel Power

C Function Prototype

```
ViStatus rssifs_readChannelPower (
      ViSession instrumentHandle,
      ViReal64* channelPower);
```

Basic Function Prototype

```
Function rssifs_readChannelPower (
      ByVal instrumentHandle As ViSession,
      channelPower As ViReal64) As ViStatus
```

Purpose

This function returns channel power measurement result computed over the trace cache data.

Note

- Channel power measurement must be enabled!
- This function is only available in the frequency domain (span > 0).

Parameters List

3. **ViSession instrumentHandle [in]**
   This control accepts the Instrument Handle returned by the Initialize function to select the desired instrument driver session.

   Default Value: None

4. **ViReal64 channelPower [out]**
   Returns result of channel power measurement in dBm.

Return Value

Returns the status code of this operation.

The meaning of the status code is described in section Error (Status) Codes.

1.7.5.7 Read Occupied Bandwidth

C Function Prototype

```
ViStatus rssifs_readOccupiedBandwidth (
      ViSession instrumentHandle,
      ViReal64[] occupiedBandwidth);
```

Basic Function Prototype

```
Function rssifs_readOccupiedBandwidth (
      ByVal instrumentHandle As ViSession,
      occupiedBandwidth As ViReal64) As ViStatus
```

Purpose

This function returns result of occupied bandwidth measurement. The occupied bandwidth is defined as the bandwidth containing a defined percentage of the total transmitted power.

Note

- Occupied bandwidth measurement must be enabled!
- This function is only available in the frequency domain (span > 0).

Parameters List

3. **ViSession instrumentHandle [in]**
   This control accepts the Instrument Handle returned by the Initialize function to select the desired instrument driver session.

   Default Value: None

4. **ViReal64[] occupiedBandwidth [out]**
   Returns result of occupied bandwidth measurement, start frequency and stop frequency of the occupied bandwidth area in Hz.

   Where results are in order: <OBW>,<start_freq>,<stop_freq>

Note

The array must contain at least 3 elements.

Return Value

Returns the status code of this operation.

The meaning of the status code is described in section Error (Status) Codes.

1.7.5.8 Read Time Domain Power

C Function Prototype	ViStatus rssifs_readTimeDomainPower (ViSession instrumentHandle, ViReal64* power);
Basic Function Prototype	Function rssifs_readTimeDomainPower (ByVal instrumentHandle As ViSession, power As ViReal64) As ViStatus
Purpose	This function returns time domain power measurement result computed over the trace cache data.
Note	▪ Time domain power measurement must be enabled! ▪ Measurement limits (configured limit lines) might apply. ▪ This function is only available in the time domain (zero span mode).
Parameters List	**3. ViSession instrumentHandle [in]** This control accepts the Instrument Handle returned by the Initialize function to select the desired instrument driver session. Default Value: None **4. ViReal64 power [out]** Returns result of time domain power measurement in dBm.
Note	The mean power or the rms power can be measured by means of the individual power values.
Return Value	Returns the status code of this operation. The meaning of the status code is described in section Error (Status) Codes.

1.7.5.9 Read Trace Data

C Function Prototype

```
ViStatus rssifs_readTraceData (
      ViSession instrumentHandle,
      ViReal64[] traceData,
      ViInt32* startIndex,
       ViInt32* samplesReturned);
```

Basic Function Prototype

```
Function rssifs_readTraceData (
      ByVal instrumentHandle As ViSession,
      traceData As ViReal64,
      startIndex As ViInt32,
      samplesReturned As ViInt32) As ViStatus
```

Purpose

This function reads out raw trace data (frequency or zero-span envelope) directly from the instrument. Trace data comprises the currently measured sweep pixels. This function does not have any influence to the cached trace data.

Note

This function need not return data of full sweep as is defined with function Configure Sweep Points (rssifs_confSweepPoints) and may return only a currently measured part of the trace data instead. Full sweep should be combined from appropriate number of sub-sweeps.

To get complete data of sweep at once use function Read Complete Sweep Data (rssifs_readCompleteSweepData).

This function is useful for high speed data measurement.

This function does not send any triggering request to the instrument. To successfully proceed, measurement must be initiated prior to call this function.

Parameters List

1. **ViSession instrumentHandle [in]**

   This control accepts the Instrument Handle returned by the Initialize function to select the desired instrument driver session.

   Default Value: None

2. **ViReal64[] traceData [out]**

   Returns the trace data.

Note

The array must contain at least as many elements as is configured with function Configure Sweep Points (rssifs_confSweepPoints).

Transferred data are returned in current trace unit. See Configure Trace Unit (rssifs_confTraceUnit) function.

3. **ViInt32 startIndex [out]**

   Returns offset of the first pixel (0 to Pixel Count - 1).

4. **ViInt32 samplesReturned [out]**

   Returns number of trace samples read out from the instrument.

Return Value

Returns the status code of this operation.

The meaning of the status code is described in section Error (Status) Codes.

1.7.5.10 Read Complete Sweep Data

C Function Prototype

```
ViStatus rssifs_readCompleteSweepData (
      ViSession instrumentHandle,
      ViReal64[] traceData,
      ViInt32* samplesReturned);
```

Basic Function Prototype

```
Function rssifs_readCompleteSweepData (
      ByVal instrumentHandle As ViSession,
      traceData As ViReal64,
      samplesReturned As ViInt32) As ViStatus
```

Purpose

This function reads out evaluated trace cache data (frequency or zero-span envelope) of completed sweep.

Trace data can be overwritten in each measurement (CLEAR/WRITE mode), averaged over several measurements (AVERAGE mode) and maximum or minimum value can be determined from several measurements (MAX HOLD or MIN HOLD) by means of Configure Trace Mode (rssifs_confTraceMode) settings.

Note

If trace data are available and number of trace points is equal to value defined by Configure Sweep Points (rssifs_confSweepPoints) function, trace data are returned. Otherwise this function waits until the full sweep is available (predicts time to complete sweep) and then returns trace data or error if data are not available.

This function does not send any triggering request to the instrument. To successfully proceed, measurement must be initiated prior to call this function.

Parameters List

1. **ViSession instrumentHandle [in]**

   This control accepts the Instrument Handle returned by the Initialize function to select the desired instrument driver session.

   Default Value: None

2. **ViReal64[] traceData [out]**

   Returns the trace data.

Note

The array must contain at least as many elements as is configured with function Configure Sweep Points (rssifs_confSweepPoints).

Transferred data are returned in current trace unit. See Configure Trace Unit (rssifs_confTraceUnit) function.

3. **ViInt32 samplesReturned [out]**

   Returns the number of trace data points.

Return Value

Returns the status code of this operation.

The meaning of the status code is described in section Error (Status) Codes.

1.7.6 Utility Functions

Description

This class of functions provides lower level functions to communicate with the instrument, and change the instrument's parameters. It also provides functions which allow the user to determine the current status of the instrument.

1.7.6.1 Time Out

Description

This class of function sets (gets) a minimum timeout value for driver I/O transactions.

1.7.6.1.1 Set Time Out

C Function Prototype

```
ViStatus rssifs_setTimeOut (
      ViSession instrumentHandle,
      ViInt32 timeout);
```

Basic Function Prototype

```
Function rssifs_setTimeOut (
      ByVal instrumentHandle As ViSession,
      ByVal timeout As ViInt32) As ViStatus
```

Purpose

Sets a minimum timeout value for driver I/O transactions in milliseconds. The timeout period may vary on computer platforms.

Parameters List

1. **ViSession instrumentHandle [in]**

   This control accepts the Instrument Handle returned by the Initialize function to select the desired instrument driver session.

   Default Value: None

2. **ViInt32 timeout [in]**

   Sets the I/O timeout for all functions in the driver. It is specified in milliseconds.

   Valid Range: > 0 ms

   Default Value: 10000 ms

Return Value

Returns the status code of this operation.

The meaning of the status code is described in section Error (Status) Codes.

1.7.6.1.2 Get Time Out

C Function Prototype

```
ViStatus rssifs_getTimeOut (
      ViSession instrumentHandle,
      ViInt32* timeout);
```

Basic Function Prototype

```
Function rssifs_getTimeOut (
      ByVal instrumentHandle As ViSession,
      timeout As ViInt32) As ViStatus
```

Purpose

Returns the timeout value for driver I/O transactions in milliseconds.

The timeout period may vary on computer platforms.

Parameters List

1. **ViSession instrumentHandle [in]**

   This control accepts the Instrument Handle returned by the Initialize function to select the desired instrument driver session.

   Default Value: None

2. **ViInt32 timeout [out]**

   Returns the timeout value for driver I/O transactions in milliseconds.

Return Value

Returns the status code of this operation.

The meaning of the status code is described in section Error (Status) Codes.

1.7.6.2 Flush Error Queue

C Function Prototype

```
ViStatus rssifs_FlushErrorQueue (
      ViSession instrumentHandle);
```

Basic Function Prototype

```
Function rssifs_FlushErrorQueue (
      ByVal instrumentHandle As ViSession) As ViStatus
```

Purpose

This function deletes internal instrument's driver error queue. It also sends a request to flush all events waiting to be read from the instrument.

Parameters List

1. **ViSession instrumentHandle [in]**

   This control accepts the Instrument Handle returned by the Initialize function to select the desired instrument driver session.

   Default Value: None

Return Value

Returns the status code of this operation.

The meaning of the status code is described in section Error (Status) Codes.

1.7.6.3 State Checking

C Function Prototype

```
ViStatus rssifs_errorCheckState (
      ViSession instrumentHandle,
      ViBoolean stateChecking);
```

Basic Function Prototype

```
Function rssifs_errorCheckState (
      ByVal instrumentHandle As ViSession,
      ByVal stateChecking As ViBoolean) As ViStatus
```

Purpose

This function enables (disables) operation(s) status checking.

⚠ **Caution**

Status checking is by default enabled. It is not recommended to disable it.

Note

When disabled, status checking is not performed and function rssifs_error_query does not provide any error message information.

When disabled, internal event handling mechanism (interrupt pipe checking & control transfer handshake checking) is also disabled.

When disabled, interaction in between instrument driver on the host computer and device's firmware is affected. Performance of the instrument driver calls might increase, but behaviour is not fully predictable (synchronization and timing mechanism is disabled).

Parameters List

1. **ViSession instrumentHandle [in]**

   This control accepts the Instrument Handle returned by the Initialize function to select the desired instrument driver session.

   Default Value: None

2. **ViBoolean stateChecking [in]**

   This control switches instrument status checking On or Off.

   Valid Range:

   - VI_FALSE (0) - Off
   - VI_TRUE (1) - On

   Default Value: VI_TRUE (1)

Return Value

Returns the status code of this operation.

The meaning of the status code is described in section Error (Status) Codes.

1.7.6.4 Warning Checking

C Function Prototype

```
ViStatus rssifs_warningCheckState (
      ViSession instrumentHandle,
      ViBoolean warningChecking);
```

Basic Function Prototype

```
Function rssifs_warningCheckState (
      ByVal instrumentHandle As ViSession,
      ByVal warningChecking As ViBoolean) As ViStatus
```

Purpose

This function enables (disables) warning checking. Warning checking is by default disabled.

Note

Warnings are produced by the instrument as asynchronous messages. When pre-enforcement validation process changed value of a facultative register the warning message is produced.

To enable this function user should be familiar with the register based communication with the instrument.

When enabled, warnings are queued in the instrument driver's error queue from where is possible get them via calling Error-Query (rssifs_error_query).

Parameters List

1. **ViSession instrumentHandle [in]**

   This control accepts the Instrument Handle returned by the Initialize function to select the desired instrument driver session.

   Default Value: None

2. **ViBoolean warningChecking [in]**

   This control switches instrument warning checking On or Off.

   Valid Range:

   - VI_FALSE (0) - Off
   - VI_TRUE (1) - On

   Default Value: VI_FALSE (0)

Return Value

Returns the status code of this operation.

The meaning of the status code is described in section Error (Status) Codes.

1.7.6.5 Reset

C Function Prototype

```
ViStatus rssifs_reset (
      ViSession instrumentHandle);
```

Basic Function Prototype

```
Function rssifs_reset (
      ByVal instrumentHandle As ViSession) As ViStatus
```

Purpose

This function resets the instrument to a known state.

Parameters List

1. **ViSession instrumentHandle [in]**

   This control accepts the Instrument Handle returned by the Initialize function to select the desired instrument driver session.

   Default Value: None

Return Value

Returns the status code of this operation.

The meaning of the status code is described in section Error (Status) Codes.

1.7.6.6 Self-Test

C Function Prototype

```
ViStatus rssifs_self_test (
      ViSession instrumentHandle,
      ViInt16* selfTestResult,
      ViChar[] selfTestMessage);
```

Basic Function Prototype

```
Function rssifs_self_test (
      ByVal instrumentHandle As ViSession,
      selfTestResult As ViInt16,
      selfTestMessage As ViChar) As ViStatus
```

Purpose

This function runs the instrument's self test routine and returns the test result(s).

Parameters List

1. **ViSession instrumentHandle [in]**

   This control accepts the Instrument Handle returned by the Initialize function to select the desired instrument driver session.

   Default Value: None

2. **ViInt16 selfTestResult [out]**

   This control contains the value returned from the instrument self-test. Zero means success.

   For any other code, see self-test mask below:

   - bit 0: SRAM
   - bit 1: IRAM
   - bit 2: FPGA
   - bit 3: DSP
   - bit 4: EEPROM AB (check-sum)
   - bit 5: EEPROM DB (check-sum)
   - bit 6: DDC
   - bit 7: Reserved
   - bit 8: Reserved
   - bit 9: TEMPERATURE AB (range check)
   - bit 10: TEMPERATURE DB (range check)
   - bit 11: TEMPERATURE OSC (range check)
   - bit 12: VOLTAGE AB (range check)
   - bit 13: VOLTAGE DB (range check)
   - bit 14: VCO
   - bit 15: DC Bias

3. **ViChar[] selfTestMessage [out]**

   This control contains the string returned from the self test.

Note The array must contain at least 256 elements ViChar[256].

Return Value

Returns the status code of this operation.

The meaning of the status code is described in section Error (Status) Codes.

1.7.6.7 Error-Query

C Function Prototype

```
ViStatus rssifs_error_query (
      ViSession instrumentHandle,
      ViInt32* errorCode,
      ViChar[] errorMessage);
```

Basic Function Prototype

```
Function rssifs_error_query (
      ByVal instrumentHandle As ViSession,
      errorCode As ViInt32,
      errorMessage As ViChar) As ViStatus
```

Purpose

This function reads an error code from the instrument driver's error queue.

Parameters List

1. **ViSession instrumentHandle [in]**

   This control accepts the Instrument Handle returned by the Initialize function to select the desired instrument driver session.

   Default Value: None

2. **ViInt32 errorCode [out]**

   This control returns the error code read from the instrument driver's error queue.

Note Error code is represented by an Event Id.

3. **ViChar[] errorMessage [out]**

   This control returns the error message string read from the instrument driver's error message queue.

Note The array must contain at least 256 elements ViChar[256].

Return Value

Returns the status code of this operation.

The meaning of the status code is described in section Error (Status) Codes.

1.7.6.8 Error Message

C Function Prototype

```
ViStatus rssifs_error_message (
      ViSession instrumentHandle,
      ViStatus statusCode,
      ViChar[] message);
```

Basic Function Prototype

```
Function rssifs_error_message (
      ByVal instrumentHandle As ViSession,
      ByVal statusCode As ViStatus,
      message As ViChar) As ViStatus
```

Purpose

This function takes the Status Code returned by the instrument driver functions, interprets it and returns it as a user readable string.

Parameters List

1. **ViSession instrumentHandle [in]**

   This control accepts the Instrument Handle returned by the Initialize function to select the desired instrument driver session.

   Default Value: VI_NULL

2. **ViStatus statusCode [in]**

   This control accepts the Status Code returned from the instrument driver functions.

   Default Value:

   - 0 - VI_SUCCESS

3. **ViChar[] message [out]**

   This control returns the interpreted Status Code as a user readable message string.

Note

The array must contain at least 256 elements ViChar[256].

Return Value

Returns the status code of this operation.

The meaning of the status code is described in section Error (Status) Codes.

1.7.6.9 Revision Query

C Function Prototype

```
ViStatus rssifs_revision_query (
      ViSession instrumentHandle,
      ViChar[] instrumentDriverRevision,
      ViChar[] firmwareRevision);
```

Basic Function Prototype

```
Function rssifs_revision_query (
      ByVal instrumentHandle As ViSession,
      instrumentDriverRevision As ViChar,
      firmwareRevision As ViChar) As ViStatus
```

Purpose

This function returns the revision numbers of the instrument driver and instrument module firmware, and tells the user with which instrument firmware this revision of the driver is compatible.

Parameters List

1. **ViSession instrumentHandle [in]**

   This control accepts the Instrument Handle returned by the Initialize function to select the desired instrument driver session.

   Default Value: None

2. **ViChar[] instrumentDriverRevision [out]**

   This control returns the Instrument Driver Software Revision.

Note The array must contain at least 256 elements ViChar[256].

3. **ViChar[] firmwareRevision [out]**

   This control returns the Series300 module firmware revision.

Note The array must contain at least 256 elements ViChar[256].

Return Value

Returns the status code of this operation.

The meaning of the status code is described in section Error (Status) Codes.

1.7.7 Close

C Function Prototype

```
ViStatus rssifs_close (
      ViSession instrumentHandle);
```

Basic Function Prototype

```
Function rssifs_close (
      ByVal instrumentHandle As ViSession) As ViStatus
```

Purpose

This function closes session to the instrument.

Note

The instrument must be reinitialized to use it again.

Parameters List

1. **ViSession instrumentHandle [in]**

   This control accepts the Instrument Handle returned by the Initialize function to select the desired instrument driver session.

   Default Value: None

Return Value

Returns the status code of this operation.

The meaning of the status code is described in section Error (Status) Codes.

1.8 Error (Status) Codes

Description

The status code either indicates success or describes an error or warning condition. You are able to examine the status code from each call to an instrument driver function to determine if an error occurred. To obtain a text description of the status code, call the **rssifs_error_message** function.

Error Queue

Error messages produced by device are queued and can be queried calling the **rssifs_error_query** function. Up to 512 messages is queued using FILO (First-In Last-Out) method. If the error queue is full, error RSSIFS_ERROR_QUEUE_OVERFLOW (0xBFFC09F8) is produced. All new upcoming errors are lost then.

Status Code

The general meaning of the status code is as follows:

Value	Meaning
0	Success
Positive Values	Warning
Negative Values	Errors

Table 0-2: The Meaning of the Status Code

Details

This instrument driver also returns errors and warnings defined by other sources. The following table defines the ranges of additional status codes that this driver can return. The table lists the different include files that contain the defined constants for the particular status codes:

Numeric Range (in Hex)	Status Code	Types
3FFF0000 to 3FFFFFFF	VISA	Warnings
3FFC0000 to 3FFCFFFF	VXIPnP	Driver Warnings
BFFF0000 to BFFFFFFF	VISA	Errors
BFFC0000 to BFFCFFFF	VXIPnP	Driver Errors

Table 0-3: Status Codes

List of all known instrument driver warnings codes:

Value	Text Description
0x3FFC0101	WARNING: ID Query not supported.
0x3FFC0102	WARNING: Reset not supported.
0x3FFC0103	WARNING: Self-test not supported.
0x3FFC0104	WARNING: Error Query not supported.
0x3FFC0105	WARNING: Revision Query not supported.
0x3FFC09F0	WARNING: Pre-enforcement validation process changed value of a facultative register.
0x3FFC09F1	WARNING: Data retrieved from trace cache.

Table 0-4: Warnings Codes

List of all known instrument driver errors codes:

Value	Text Description
0xBFFC0001	ERROR: Parameter 1 out of range.
0xBFFC0002	ERROR: Parameter 2 out of range.
0xBFFC0003	ERROR: Parameter 3 out of range.
0xBFFC0004	ERROR: Parameter 4 out of range.
0xBFFC0005	ERROR: Parameter 5 out of range.
0xBFFC0006	ERROR: Parameter 6 out of range.
0xBFFC0007	ERROR: Parameter 7 out of range.
0xBFFC0008	ERROR: Parameter 8 out of range.
0xBFFC0011	ERROR: Identification query failed.
0xBFFC0012	ERROR: Interpreting instrument response.
0xBFFC0800	ERROR: Opening the specified file.
0xBFFC0801	ERROR: Writing to the specified file.
0xBFFC0803	ERROR: Interpreting the instrument's response.
0xBFFC0809	ERROR: Parameter 9 out of range.
0xBFFC080A	ERROR: Parameter 10 out of range.
0xBFFC080B	ERROR: Parameter 11 out of range.
0xBFFC080C	ERROR: Parameter 12 out of range.
0xBFFC080D	ERROR: Parameter 13 out of range.
0xBFFC080E	ERROR: Parameter 14 out of range.
0xBFFC080F	ERROR: Parameter 15 out of range.
0xBFFC09F0	ERROR: Instrument status error.
0xBFFC09F1	ERROR: Instrument configuration error.
0xBFFC09F2	ERROR: Required instrument's option is not installed.
0xBFFC09F3	ERROR: Required instrument model is not connected.
0xBFFC09F4	ERROR: Selected register name is not supported by the instrument.
0xBFFC09F5	ERROR: Invalid value (value out of range).
0xBFFC09F6	ERROR: Range table for selected register name is not available.
0xBFFC09F7	ERROR: NULL pointer passed as parameter.
0xBFFC09F8	ERROR: The error queue is overflowed.
0xBFFC09F9	ERROR: Data not available.
0xBFFC09FA	ERROR: Data are corrupted.
0xBFFC09FB	ERROR: Settings conflict. Passed parameter does not match with the current instrument's settings.
0xBFFC09FC	ERROR: Function or parameter is for any reason reserved.
0xBFFC09FD	ERROR: Current measurement has been aborted.
0xBFFC09FE	ERROR: Error on execution of the function.

Table 0-5: Error Codes

1.9 Execution Timeout

Description

Timeout value specifies the minimum time to use (in milliseconds) when accessing the device associated with the given session. A timeout value means that operations should wait for the device to respond at the least defined amount of time. The timeout value is set via **rssifs_setTimeOut** function. The actual timeout value is retrieved via **rssifs_getTimeOut** function.

Note

Notice that the actual timeout value used by the driver may be higher than the requested one.

1.10 Alphabetical List of Functions

Contacts

List of your Rohde&Schwarz Partners

- For comprehensive information about Rohde&Schwarz, please visit our Rohde&Schwarz Homepage (http://www.rohde-schwarz.com/).
- For queries regarding technical aspects of our products, please contact our Customer Support (http://www.rohde-schwarz.com/www/dev_center.nsf/html/service_customerservicehotline).
- For international services, please contact our Service Partners (http://www.services.rohde-schwarz.com/).
- For information on training and seminars, please visit our Training Center (http://www.training.rohde-schwarz.com/).
- For latest instrument driver updates, please see our Rohde&Schwarz Drivers (http://www.rohde-schwarz.com/www/download.nsf/driver_frameset).

Customer upport center

USA & Canada
Monday to Friday (except US-state holidays)
8:00 AM – 8:00 PM Eastern Standard Time (EST)
USA: 888-test-rsa (888-837-8772) (opt 2)
From outside USA: +1 410 910 7800 (opt 2)
Fax: 410 910 7801
E-Mail: Customer.Support@rsa.rohde-schwarz.com

Rest of World
Monday to Friday (except German-state holidays)
08:00 – 17:00 Central European Time (CET)
Europe: +49 (0) 180 512 42 42
From outside Europe: +49 89 4129 13776
Fax: +49 (0) 89 41 29 637 78
E-Mail: CustomerSupport@rohde-schwarz.com

1.11 Remote Control Programming Examples

Description

All the programming examples are written in ANSI C language. Examples are commented; execution results are appended after the source code.

FS300 Spectrum Analyzer input is provided with the external source of RF signal:

- Frequency: 1GHz
- Level: -30 dBm
- Modulation: AM switched when needed

Note

The trace data are displayed using embedded illustration pictures, where the pictures are not part of the examples. They have been created with external software.

The timings printed out with the examples may vary on different systems, depending on the system speed and load.

1.11.1 Error Handling & Time Profiling

1.11.1.1 Source Code

```
/*****************************************************************************
 *
 * Title:   Error Handling & Time Profiling
 *
 * Purpose: This example shows basic principles of error handling.
 *
 *****************************************************************************/

#include <ansi_c.h>
#include "rssifs.h"

/*** Macros & definitions ***************************************************/

#define CHECKERR( fCal ) \
    { \
        fCalStartTime = clock();\
        error = (fCal);\
        status = error;\
        printf(" Line %ld (%ld ms): %s\n", __LINE__,\
            (fCalTime = (clock() - fCalStartTime)), #fCal);\
        if (error != VI_SUCCESS)\
            {\
            rssifs_error_message (io, error, error_message);\
            printf("\tFunction Call Status: 0x%08X, %s\n", error, error_message);\
            rssifs_error_query(io, &error, error_message);\
            printf("\tInstrument Error: 0x%X, %s\n", error, error_message);\
            }\
    }

#define RESOURCE_NAME   "USB::0xAAD::0x6::100202"   // Resource name

/*** Main *******************************************************************/

int main (int argc, char *argv[])
{
    ViStatus    error               = VI_SUCCESS,
                status              = VI_SUCCESS;
    clock_t     fCalTime            = 0,
                fCalStartTime       = 0;
    ViChar      error_message[256];
    ViSession   io;

    ViReal64    sweep_time          = 0.0;

    /* Define remote connection parameters (as default values) */

    ViBoolean   IDQuery             = VI_TRUE,
                resetDevice         = VI_TRUE;
    ViRsrc      resourceName        = RESOURCE_NAME;

    CHECKERR (rssifs_init (resourceName, IDQuery, resetDevice, &io));

    printf (

    "\n --- Error Checking ---\n\n"
    "\tThis example uses macro CHECKERR to cover error handling.\n"
```

```
"\tBasic principle of error handling is as follows:\n\n"

"\t(1) status code (status) is returned by function call (fCal)\n"
"\t(2) if status code is <> VI_SUCCESS (0):\n"
"\t    - translate status code to message using rssifs_error_message"
" function\n"
"\t    - call rssifs_error_query function to get more information from error"
" queue\n"
"\t(3) if status code is equal to VI_SUCCESS (0):\n"
"\t    - function call succeed\n\n"

);

/* Correct function call */
CHECKERR (rssifs_confStartStopFrq (io, 0, 0.0, 3.0e9));
/* Passed wrong data to generate error */
CHECKERR (rssifs_confStartStopFrq (io, 0, 3.0e9, 0.0));

printf (

"\n --- Warning Checking ---\n\n"
"\tWarning checking provides additional information of device operations.\n"
"\tWarnings are produced by the instrument as asynchronous messages.\n"
"\tIt can be enabled using function rssifs_warningCheckState.\n"
"\tHandling of the warnings is the same as for errors.\n\n"

);

CHECKERR (rssifs_warningCheckState (io, VI_TRUE));
/* Try to produce warnings ... */
CHECKERR (rssifs_confSweep (io, 1, 0));
CHECKERR (rssifs_confSpanCenterFrq (io, 0, 0.0, 1.0e9));
CHECKERR (rssifs_getSweepTime (io, &sweep_time));
CHECKERR (rssifs_warningCheckState (io, VI_FALSE));

printf (

"\n --- Disable Warning & Error Checking ---\n\n"
"\tWarning & Error checking can be disabled by calling function"
" rssifs_errorCheckState.\n"
"\tSee description:\n\n"

"\t(1) When disabled, status checking is not performed and function\n"
"\t    rssifs_error_query do not provide any error message\n"
"\t    information.\n"

"\t(2) When disabled, internal event handling mechanism (interrupt\n"
"\t    pipe checking & control transfer handshake checking) is also\n"
"\t    disabled.\n"

"\t(3) When disabled, interaction in between instrument driver on\n"
"\t    the host computer and device's firmware is affected. Performance\n"
"\t    of the instrument driver calls might increase, but behavior is\n"
"\t    not fully predictable (synchronization and timing mechanism is\n"
"\t    disabled).\n\n"

);

printf (

"\n --- Performance Improvement ---\n\n"

"\tWhen the error checking is disabled, performance might increase:\n\n"
```

```
    );

    CHECKERR (rssifs_confStartStopFrq (io, 0, 0.0, 3.0e9));
    CHECKERR (rssifs_confStartStopFrq (io, 0, 0.0, 3.0e9));
    CHECKERR (rssifs_errorCheckState (io, VI_FALSE));
    CHECKERR (rssifs_confStartStopFrq (io, 0, 0.0, 3.0e9));
    CHECKERR (rssifs_confStartStopFrq (io, 0, 0.0, 3.0e9));

    printf (

    "\n\tWARNING: Use rssifs_errorCheckState function with care!\n\n"

    );

    CHECKERR (rssifs_close (io));

    return 0;
}
```

1.11.1.2 Execution Result

```
Line 51 (1597 ms): rssifs_init (resourceName, IDQuery, resetDevice, &io)

 --- Error Checking ---

      This example uses macro CHECKERR to cover error handling.
      Basic principle of error handling is as follows:

      (1) status code (status) is returned by function call (fCal)
      (2) if status code is <> VI_SUCCESS (0):
          - translate status code to message using rssifs_error_message function
          - call rssifs_error_query function to get more information from error queue
      (3) if status code is equal to VI_SUCCESS (0):
          - function call succeed

 Line 71 (244 ms): rssifs_confStartStopFrq (io, 0, 0.0, 3.0e9)
 Line 73 (0 ms): rssifs_confStartStopFrq (io, 0, 3.0e9, 0.0)
      Function Call Status: 0xBFFC09FB, ERROR: Settings conflict. Passed parameter does
      not match with the current instrument's settings.
      Instrument Error: 0x0, No error.

 --- Warning Checking ---

      Warning checking provides additional information of device operations.
      Warnings are produced by the instrument as asynchronous messages.
      It can be enabled using function rssifs_warningCheckState.
      Handling of the warnings is the same as for errors.

 Line 85 (0 ms): rssifs_warningCheckState (io, VI_TRUE)
 Line 87 (475 ms): rssifs_confSweep (io, 1, 0)
      Function Call Status: 0x3FFC09F0, WARNING: Pre-enforcement validation process
      changed value of a facultative register.
      Instrument Error: 0x6040, WARNING: Pre-enforcement validation process changed
      value of a 'NCO_K_FACTOR' facultative register (lParam = 0x0, wParam = 0x2083).
 Line 88 (236 ms): rssifs_confSpanCenterFrq (io, 0, 0.0, 1.0e9)
 Line 89 (112 ms): rssifs_getSweepTime (io, &sweep_time)
      Function Call Status: 0x3FFC09F0, WARNING: Pre-enforcement validation process
      changed value of a facultative register.
      Instrument Error: 0x6040, WARNING: Pre-enforcement validation process changed
      value of a 'NCO_K_FACTOR' facultative register (lParam = 0x0, wParam = 0x2083).
 Line 90 (0 ms): rssifs_warningCheckState (io, VI_FALSE)
```

```
--- Disable Warning & Error Checking ---

     Warning & Error checking can be disabled by calling function
     rssifs_errorCheckState.

     See description:

     (1) When disabled, status checking is not performed and function
         rssifs_error_query do not provide any error message
         information.
     (2) When disabled, internal event handling mechanism (interrupt
         pipe checking & control transfer handshake checking) is also
         disabled.
     (3) When disabled, interaction in between instrument driver on
         the host computer and device's firmware is affected. Performance
         of the instrument driver calls might increase, but behavior is
         not fully predictable (synchronization and timing mechanism is
         disabled).

--- Performance Improvement ---

     When the error checking is disabled, performance might increase:

Line 123 (196 ms): rssifs_confStartStopFrq (io, 0, 0.0, 3.0e9)
Line 124 (136 ms): rssifs_confStartStopFrq (io, 0, 0.0, 3.0e9)
Line 125 (0 ms): rssifs_errorCheckState (io, VI_FALSE)
Line 126 (7 ms): rssifs_confStartStopFrq (io, 0, 0.0, 3.0e9)
Line 127 (6 ms): rssifs_confStartStopFrq (io, 0, 0.0, 3.0e9)

     WARNING: Use rssifs_errorCheckState function with care!

Line 135 (0 ms): rssifs_close (io)
```

1.11.2 Measurement in Frequency Domain

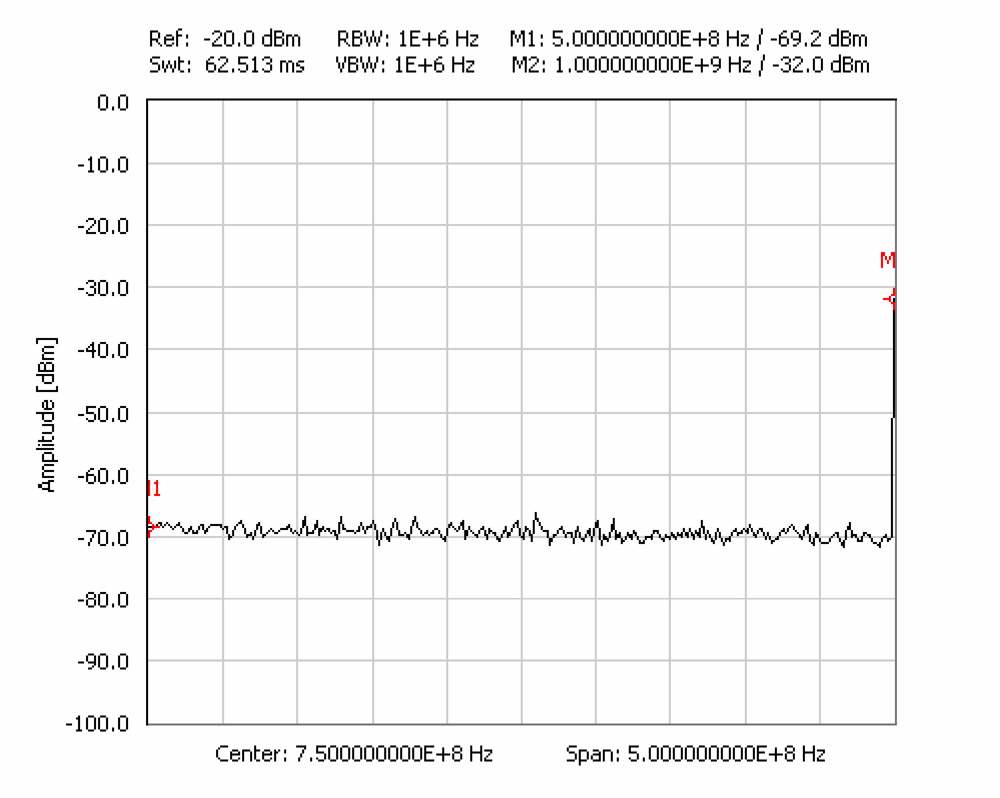
1.11.2.1 Source Code

```
/*****************************************************************************
 *
 * Title:   Measurement in Frequency Domain
 *
 * Purpose: This example shows how to setup measurement in frequency domain.
 *
 *****************************************************************************/

#include <ansi_c.h>
#include "rssifs.h"

/*** Macros & definitions ***************************************************/

#define CHECKERR( fCal ) \
    { \
        fCalStartTime = clock();\
        error = (fCal);\
        status = error;\
        printf(" Line %ld (%ld ms): %s\n", __LINE__,\
            (fCalTime = (clock() - fCalStartTime)), #fCal);\
        if (error != VI_SUCCESS)\
            {\
            rssifs_error_message (io, error, error_message);\
            printf("\tFunction Call Status: 0x%08X, %s\n", error, error_message);\
            rssifs_error_query(io, &error, error_message);\
            printf("\tInstrument Error: 0x%X, %s\n", error, error_message);\
            }\
    }

#define RESOURCE_NAME   "USB::0xAAD::0x6::100202"   // Resource name

/*** Main *******************************************************************/

int main (int argc, char *argv[])
{
    ViStatus    error               = VI_SUCCESS,
                status              = VI_SUCCESS;
    clock_t     fCalTime            = 0,
                fCalStartTime       = 0;
    ViChar      error_message[256];
    ViSession   io;

    ViReal64    startFrequency      = 500.0e6,
                stopFrequency       = 1.0e9;

    /* Define remote connection parameters (as default values) */

    ViBoolean   IDQuery             = VI_TRUE,
                resetDevice         = VI_TRUE;
    ViRsrc      resourceName        = RESOURCE_NAME;

    CHECKERR (rssifs_init (resourceName, IDQuery, resetDevice, &io));

    printf (

    "\n --- Measurement in Frequency Domain ------------------------------------\n\n"
    "\tRF Generator is connected to the analyzer's input\n"
    "\t- Frequency: 1 GHz\n"
    "\t- Level: -30.0 dBm\n"
    "\t- Modulation: none\n"
    "\n ------------------------------------------------------------------------\n\n"

    );

    /* Set reference level */
    CHECKERR (rssifs_confRefLevel (io, -20.0));
    /* Set start and stop frequency */
    CHECKERR (rssifs_confStartStopFrq (io, 0, startFrequency, stopFrequency));

    /* Read trace data of completed sweep */
    {
        ViInt32     samplesReturned = 0;
        ViReal64    traceData[250];

        CHECKERR (rssifs_readCompleteSweepData (io, traceData, &samplesReturned));

        if (error == VI_SUCCESS)
            {
            printf ("\tSamples Returned: %ld\n"
                    "\tTrace Data [Vrms]: %.3Le, %.3Le, %.3Le ...\n"
                    "\tStart Frequency [Hz]: %.3Le\n"
                    "\tStop Frequency [Hz]: %.3Le\n"
                    "\tFrequency Step [Hz]: %.3Le\n",
                    samplesReturned,
                    traceData[0], traceData[1], traceData[2],
                    startFrequency,
                    stopFrequency,
                    (stopFrequency - startFrequency) / samplesReturned);
            }
    }

    CHECKERR (rssifs_close (io));

    return 0;
}
```

1.11.2.2 Execution Results

```
Line 52 (1489 ms): rssifs_init (resourceName, IDQuery, resetDevice, &io)

 --- Measurement in Frequency Domain ------------------------------------

      RF Generator is connected to the analyzer's input
      - Frequency: 1 GHz
      - Level: -30.0 dBm
      - Modulation: none

 ------------------------------------------------------------------------

 Line 66 (38 ms): rssifs_confRefLevel (io, -20.0)
 Line 68 (136 ms): rssifs_confStartStopFrq (io, 0, startFrequency, stopFrequency)
 Line 75 (112 ms): rssifs_readCompleteSweepData (io, traceData, &samplesReturned)
      Samples Returned: 250
      Trace Data [Vrms]: 7.422e-05, 8.586e-05, 9.623e-05 ...
      Start Frequency [Hz]: 5.000e+08
      Stop Frequency [Hz]: 1.000e+09
      Frequency Step [Hz]: 2.000e+06
 Line 92 (0 ms): rssifs_close (io)
```

1.11.2.3 Display Results



Measurement in Frequency Domain

1.11.3 Measurement in Time Domain

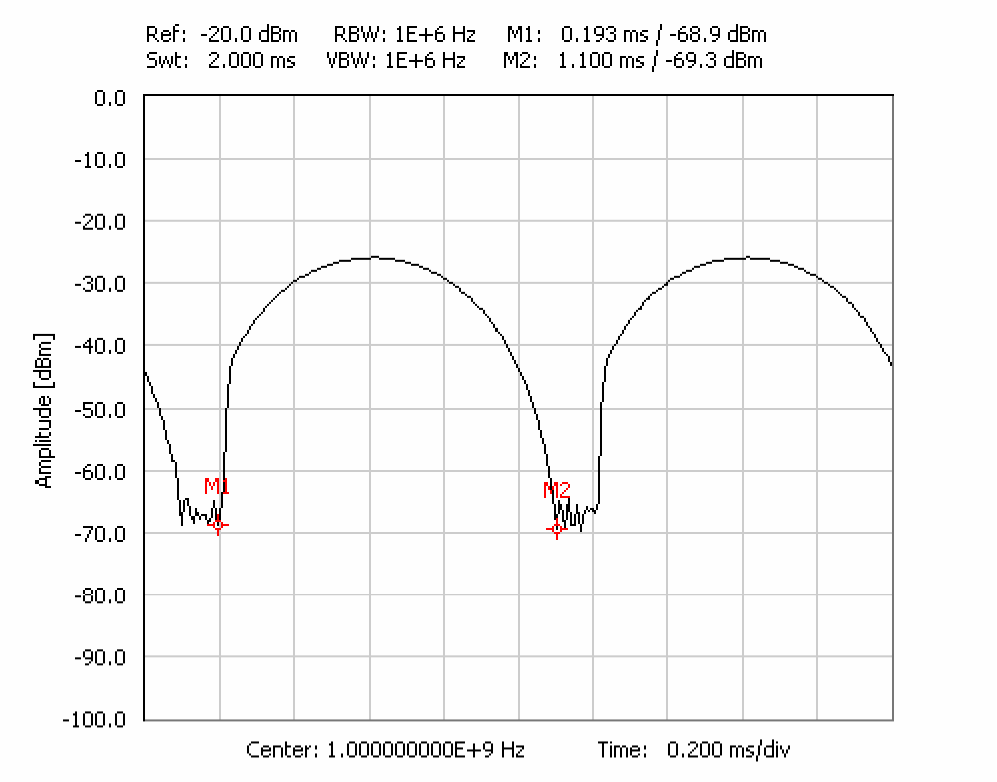
1.11.3.1 Source Code

```
/****************************************************************************
 *
 * Title:   Measurement in Time Domain
 *
 * Purpose: This example shows how to setup measurement in time domain
 *          (Zero span measurement).
 *
 ****************************************************************************/

#include <ansi_c.h>
#include "rssifs.h"

/*** Macros & definitions **************************************************/

#define CHECKERR( fCal ) \
    { \
        fCalStartTime = clock();\
        error = (fCal);\
        status = error;\
        printf(" Line %ld (%ld ms): %s\n", __LINE__,\
            (fCalTime = (clock() - fCalStartTime)), #fCal);\
        if (error != VI_SUCCESS)\
            {\
            rssifs_error_message (io, error, error_message);\
            printf("\tFunction Call Status: 0x%08X, %s\n", error, error_message);\
            rssifs_error_query(io, &error, error_message);\
            printf("\tInstrument Error: 0x%X, %s\n", error, error_message);\
            }\
    }

#define RESOURCE_NAME   "USB::0xAAD::0x6::100202"   // Resource name

/*** Main ******************************************************************/

int main (int argc, char *argv[])
{
    ViStatus    error               = VI_SUCCESS,
                status              = VI_SUCCESS;
    clock_t     fCalTime            = 0,
                fCalStartTime       = 0;
    ViChar      error_message[256];
    ViSession   io;

    /* Define remote connection parameters (as default values) */

    ViBoolean   IDQuery             = VI_TRUE,
                resetDevice         = VI_TRUE;
    ViRsrc      resourceName        = RESOURCE_NAME;

    CHECKERR (rssifs_init (resourceName, IDQuery, resetDevice, &io));

    printf (

    "\n --- Measurement in Time Domain ----------------------------------------\n\n"
    "\tRF Generator is connected to the analyzer's input\n"
    "\t- Frequency: 1 GHz\n"
    "\t- Level: -30.0 dBm\n"
    "\t- Modulation: AM, modulation frequency 1 kHz, modulation depth 100 %%\n"
    "\n ------------------------------------------------------------------------\n\n"

    );

    /* Set reference level */
    CHECKERR (rssifs_confRefLevel (io, -20.0));
    /* Set center frequency and span */
    CHECKERR (rssifs_confSpanCenterFrq (io, 0, 0.0, 1.0e9));
    /* Set RBW = 1MHz and VBW = 1MHz */
    CHECKERR (rssifs_configureBandwidth (io, 16, 21));
    /* Configure Sweep Time to 2 ms */
    CHECKERR (rssifs_confSweepTime (io, 0.002));

    /* Read trace data of completed sweep */
    {
        ViInt32     samplesReturned = 0;
        ViReal64    sweepTime       = 0.0,
                    traceData[2048];

        CHECKERR (rssifs_readCompleteSweepData (io, traceData, &samplesReturned));

        if (error == VI_SUCCESS)
            {
            CHECKERR (rssifs_getSweepTime (io, &sweepTime));

            printf ("\tSamples Returned: %ld\n"
                    "\tTrace Data [Vrms]: %.3Le, %.3Le, %.3Le ...\n"
                    "\tStart Time [s]: %Lf\n"
                    "\tStop Time [s]: %Lf\n"
                    "\tTime Step [s]: %.3Le\n",
                    samplesReturned,
                    traceData[0], traceData[1], traceData[2],
                    0.0,
                    sweepTime,
                    (sweepTime / samplesReturned));
            }
    }

    CHECKERR (rssifs_close (io));

    return 0;
}
```

1.11.3.2 Execution Result

```
Line 50 (1404 ms): rssifs_init (resourceName, IDQuery, resetDevice, &io)

 --- Measurement in Time Domain ---------------------------------------

      RF Generator is connected to the analyzer's input
      - Frequency: 1 GHz
      - Level: -30.0 dBm
      - Modulation: AM, modulation frequency 1 kHz, modulation depth 100 %

 ------------------------------------------------------------------------

 Line 64 (38 ms): rssifs_confRefLevel (io, -20.0)
 Line 66 (152 ms): rssifs_confSpanCenterFrq (io, 0, 0.0, 1.0e9)
 Line 68 (55 ms): rssifs_configureBandwidth (io, 16, 21)
 Line 70 (504 ms): rssifs_confSweepTime (io, 0.002)
 Line 78 (80 ms): rssifs_readCompleteSweepData (io, traceData, &samplesReturned)
 Line 82 (24 ms): rssifs_getSweepTime (io, &sweepTime)
      Samples Returned: 250
      Trace Data [Vrms]: 1.111e-02, 1.121e-02, 1.122e-02 ...
      Start Time [s]: 0.000000
      Stop Time [s]: 0.002000
      Time Step [s]: 8.000e-06
 Line 97 (0 ms): rssifs_close (io)
```

1.11.3.3 Display Results



Measurement in Time Domain

1.11.4 Triggered Measurement (Single Sweep Mode)

1.11.4.1 Source Code

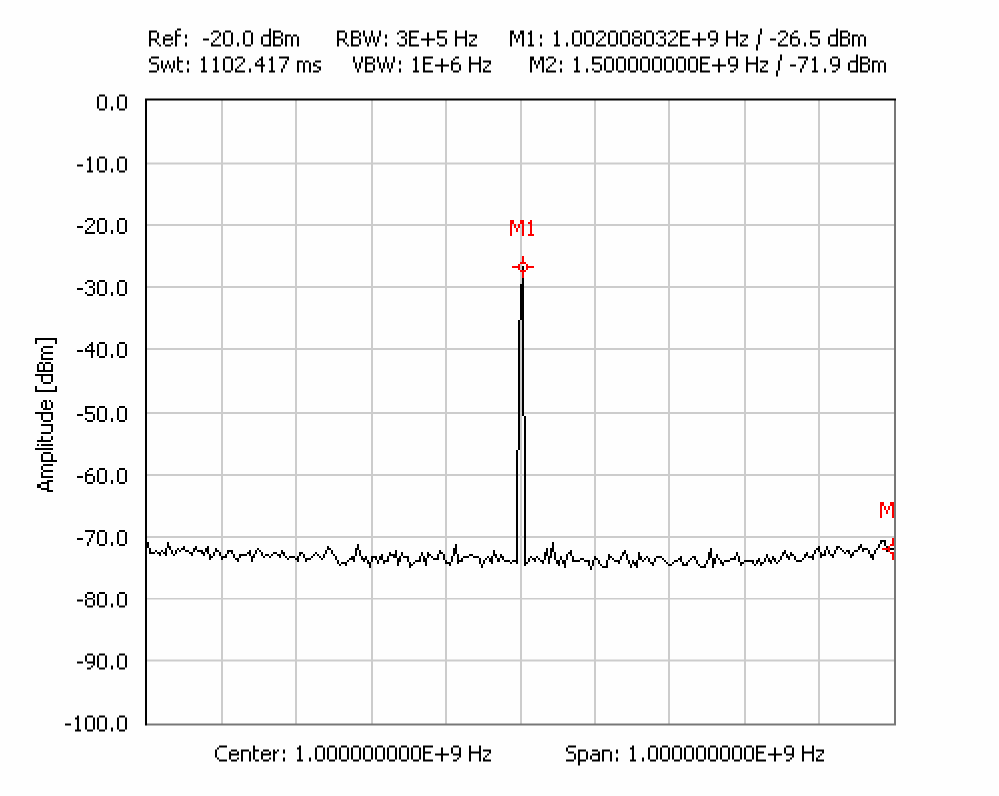
```
/*****************************************************************************
 *
 * Title:   Triggered Measurement (Single sweep mode)
 *
 * Purpose: This example shows how to setup triggered measurement.
 *
 *****************************************************************************/

#include <ansi_c.h>
#include "rssifs.h"

/*** Macros & definitions ***************************************************/

#define CHECKERR( fCal ) \
    { \
        fCalStartTime = clock();\
        error = (fCal);\
        status = error;\
        printf(" Line %ld (%ld ms): %s\n", __LINE__,\
            (fCalTime = (clock() - fCalStartTime)), #fCal);\
        if (error != VI_SUCCESS)\
            {\
            rssifs_error_message (io, error, error_message);\
            printf("\tFunction Call Status: 0x%08X, %s\n", error, error_message);\
            rssifs_error_query(io, &error, error_message);\
            printf("\tInstrument Error: 0x%X, %s\n", error, error_message);\
            }\
    }

#define RESOURCE_NAME   "USB::0xAAD::0x6::100202"   // Resource name
#define SWEEPS          10                          // Number of sweeps

/*** Main *******************************************************************/

int main (int argc, char *argv[])
{
    ViStatus    error               = VI_SUCCESS,
                status              = VI_SUCCESS;
    clock_t     fCalTime            = 0,
                fCalStartTime       = 0;
    ViChar      error_message[256];
    ViSession   io;

    ViReal64    startFrequency      = 500.0e6,
                stopFrequency       = 1.500e9,
                sweep_time          = 1.0,
                traceData[250];

    ViInt32     sweep_count         = 0,
                samplesReturned     = 0;

    /* Define remote connection parameters (as default values) */

    ViBoolean   IDQuery             = VI_TRUE,
                resetDevice         = VI_TRUE;
    ViRsrc      resourceName        = RESOURCE_NAME;
    CHECKERR (rssifs_init (resourceName, IDQuery, resetDevice, &io));

    printf (

    "\n --- Triggered Measurement (Single sweep mode) ---------------------------\n\n"
    "\tRF Generator is connected to the analyzer's input\n"
    "\t- Frequency: 1 GHz\n"
    "\t- Level: -30.0 dBm\n"
    "\t- Modulation: AM, modulation frequency 1 kHz, modulation depth 100 %%\n"
    "\n -------------------------------------------------------------------------\n\n"

    );

    /* Set reference level */
    CHECKERR (rssifs_confRefLevel (io, -20.0));
    /* Set start and stop frequency */
    CHECKERR (rssifs_confStartStopFrq (io, 0, startFrequency, stopFrequency));
    /* Set RBW to 300 kHz VBW to 1 MHz */
    CHECKERR (rssifs_configureBandwidth (io, 14, 21));

    /* Configure single sweep mode with defined number of sweeps to be executed */
    CHECKERR (rssifs_confSweep (io, 2, SWEEPS));
    /* Configure Sweep Time */
    CHECKERR (rssifs_confSweepTime (io, 1.0));
    CHECKERR (rssifs_getSweepTime (io, &sweep_time));
    printf ("\tSweep time is adjusted to %.3Lf s\n", sweep_time);

/* --- Single Sweep -------------------------------------------------------- */

    printf ("\n --- 1st triggering method: Send Trigger and Wait for operation
completed\n\n");
    CHECKERR (rssifs_actSendTrgWopc (io, 20000));
    sweep_count = 0;
    /* Read trace data */
    CHECKERR (rssifs_readCompleteSweepData (io, traceData, &samplesReturned));

    printf (

    "\n\tWarning code informs you that sweep data are after trigger\n"
    "\tis executed stored to the driver cache and function\n"
    "\trssifs_readCompleteSweepData is not reading data from instrument.\n\n"

    );

    /* Get number of measured sweeps */
    CHECKERR (rssifs_getSweepCount (io, &sweep_count));
    printf ("\tNumber of measured sweeps: %ld\n", sweep_count);

    printf ("\n --- 2nd triggering method: Send Trigger\n\n");
    CHECKERR (rssifs_actSendTrg (io));
    sweep_count = 0;
    while (sweep_count < SWEEPS)
        {
        /* Read trace data */
        CHECKERR (rssifs_readCompleteSweepData (io, traceData, &samplesReturned));
        /* Get number of measured sweeps */
        CHECKERR (rssifs_getSweepCount (io, &sweep_count));
        printf ("\tNumber of measured sweeps: %ld\n", sweep_count);
        }

/* --- Free Run ------------------------------------------------------------ */

    printf ("\n --- Triggering in the free run mode\n\n");
    /* Zero span mode */
    CHECKERR (rssifs_confSpanCenterFrq (io, 0, 0.0, startFrequency +
        (stopFrequency - startFrequency) / 2));
    /* Configure Sweep Time */
    CHECKERR (rssifs_confSweepTime (io, 0.001));
    CHECKERR (rssifs_getSweepTime (io, &sweep_time));
    printf ("\tSweep time is adjusted to %.3Lf s\n", sweep_time);

    /* Configure free run mode */
    CHECKERR (rssifs_confTrg (io, 0, -60.0, VI_FALSE));

    /* Measurement is (re)started under configured trigger and sweep conditions */
    CHECKERR (rssifs_actSendTrgWopc (io, 5000));
    /* Read trace data */
    CHECKERR (rssifs_readCompleteSweepData (io, traceData, &samplesReturned));

/* --- Video Trigger ------------------------------------------------------- */

    printf ("\n --- Video Trigger (trigger level -40 dBm, trigger offset is zero)\n\n");

    /* Configure Trigger */
    CHECKERR (rssifs_confTrg (io, 1, -40.0, VI_FALSE));
    /* Trigger Offset (trigger delay) */
    CHECKERR (rssifs_confTrgDelay (io, 0.0));

    /* Measurement is (re)started under configured trigger and sweep conditions */
    CHECKERR (rssifs_actSendTrgWopc (io, 5000));
    /* Read trace data */
    printf ("\n\tData are returned only if the trigger condition meets\n\n");
    CHECKERR (rssifs_readCompleteSweepData (io, traceData, &samplesReturned));

    printf ("\n --- Video Trigger (trigger level -40 dBm, offset is -0.2 ms)\n\n");

    /* Configure Trigger */
    CHECKERR (rssifs_confTrg (io, 1, -40.0, VI_FALSE));
    /* Trigger Offset (trigger delay) */
    CHECKERR (rssifs_confTrgDelay (io, -0.2e-3));

    /* Measurement is (re)started under configured trigger and sweep conditions */
    CHECKERR (rssifs_actSendTrgWopc (io, 5000));
    /* Read trace data */
    printf ("\n\tData are returned only if the trigger condition meets\n\n");
    CHECKERR (rssifs_readCompleteSweepData (io, traceData, &samplesReturned));

    printf ("\n");

    CHECKERR (rssifs_close (io));

    return 0;
}
```

1.11.4.2 Execution Results

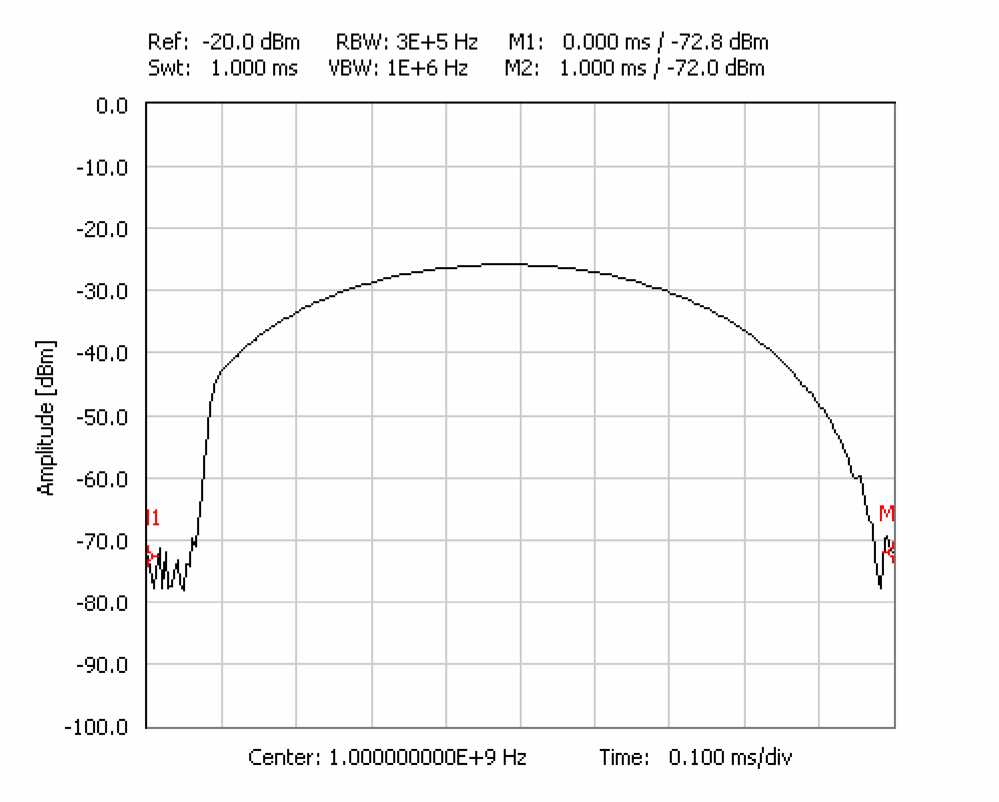
```
Line 58 (1426 ms): rssifs_init (resourceName, IDQuery, resetDevice, &io)

 --- Triggered Measurement (Single sweep mode) --------------------------

      RF Generator is connected to the analyzer's input
      - Frequency: 1 GHz
      - Level: -30.0 dBm
      - Modulation: AM, modulation frequency 1 kHz, modulation depth 100 %

 -----------------------------------------------------------------------

 Line 72 (36 ms): rssifs_confRefLevel (io, -20.0)
 Line 74 (152 ms): rssifs_confStartStopFrq (io, 0, startFrequency, stopFrequency)
 Line 76 (56 ms): rssifs_configureBandwidth (io, 14, 21)
 Line 79 (414 ms): rssifs_confSweep (io, 2, SWEEPS)
 Line 81 (496 ms): rssifs_confSweepTime (io, 1.0)
 Line 82 (192 ms): rssifs_getSweepTime (io, &sweep_time)
      Sweep time is adjusted to 1.102 s

 --- 1st triggering method: Send Trigger and Wait for operation completed

 Line 88 (11917 ms): rssifs_actSendTrgWopc (io, 20000)
 Line 91 (0 ms): rssifs_readCompleteSweepData (io, traceData, &samplesReturned)
      Function Call Status: 0x3FFC09F1, WARNING: Data retrieved from trace cache.
      Instrument Error: 0x0, No error.

      Warning code informs you that sweep data are after trigger
      is executed stored to the driver cache and function
      rssifs_readCompleteSweepData is not reading data from instrument.

 Line 102 (16 ms): rssifs_getSweepCount (io, &sweep_count)
      Number of measured sweeps: 10

 --- 2nd triggering method: Send Trigger

 Line 106 (30 ms): rssifs_actSendTrg (io)
 Line 111 (1160 ms): rssifs_readCompleteSweepData (io, traceData, &samplesReturned)
 Line 113 (8 ms): rssifs_getSweepCount (io, &sweep_count)
      Number of measured sweeps: 1
 Line 111 (1175 ms): rssifs_readCompleteSweepData (io, traceData, &samplesReturned)
 Line 113 (7 ms): rssifs_getSweepCount (io, &sweep_count)
      Number of measured sweeps: 2
 Line 111 (1232 ms): rssifs_readCompleteSweepData (io, traceData, &samplesReturned)
 Line 113 (60 ms): rssifs_getSweepCount (io, &sweep_count)
      Number of measured sweeps: 3
 Line 111 (1166 ms): rssifs_readCompleteSweepData (io, traceData, &samplesReturned)
 Line 113 (8 ms): rssifs_getSweepCount (io, &sweep_count)
      Number of measured sweeps: 4
 Line 111 (1168 ms): rssifs_readCompleteSweepData (io, traceData, &samplesReturned)
 Line 113 (8 ms): rssifs_getSweepCount (io, &sweep_count)
      Number of measured sweeps: 5
 Line 111 (1176 ms): rssifs_readCompleteSweepData (io, traceData, &samplesReturned)
 Line 113 (8 ms): rssifs_getSweepCount (io, &sweep_count)
      Number of measured sweeps: 6
 Line 111 (1168 ms): rssifs_readCompleteSweepData (io, traceData, &samplesReturned)
 Line 113 (14 ms): rssifs_getSweepCount (io, &sweep_count)
      Number of measured sweeps: 7
 Line 111 (1162 ms): rssifs_readCompleteSweepData (io, traceData, &samplesReturned)
 Line 113 (8 ms): rssifs_getSweepCount (io, &sweep_count)
```

```
     Number of measured sweeps: 8
Line 111 (1176 ms): rssifs_readCompleteSweepData (io, traceData, &samplesReturned)
Line 113 (8 ms): rssifs_getSweepCount (io, &sweep_count)
     Number of measured sweeps: 9
Line 111 (1176 ms): rssifs_readCompleteSweepData (io, traceData, &samplesReturned)
Line 113 (8 ms): rssifs_getSweepCount (io, &sweep_count)
     Number of measured sweeps: 10

--- Triggering in the free run mode

Line 123 (72 ms): rssifs_confSpanCenterFrq (io, 0, 0.0, startFrequency + (stopFrequency
     - startFrequency) / 2)
Line 125 (152 ms): rssifs_confSweepTime (io, 0.001)
Line 126 (112 ms): rssifs_getSweepTime (io, &sweep_time)
     Sweep time is adjusted to 0.001 s
Line 130 (40 ms): rssifs_confTrg (io, 0, -60.0, VI_FALSE)
Line 133 (1423 ms): rssifs_actSendTrgWopc (io, 5000)
Line 135 (0 ms): rssifs_readCompleteSweepData (io, traceData, &samplesReturned)
     Function Call Status: 0x3FFC09F1, WARNING: Data retrieved from trace cache.
     Instrument Error: 0x0, No error.

--- Video Trigger (trigger level -40 dBm, trigger offset is zero)

Line 142 (78 ms): rssifs_confTrg (io, 1, -40.0, VI_FALSE)
Line 144 (39 ms): rssifs_confTrgDelay (io, 0.0)
Line 147 (1445 ms): rssifs_actSendTrgWopc (io, 5000)

     Data are returned only if the trigger condition meets

Line 150 (0 ms): rssifs_readCompleteSweepData (io, traceData, &samplesReturned)
     Function Call Status: 0x3FFC09F1, WARNING: Data retrieved from trace cache.
     Instrument Error: 0x0, No error.

--- Video Trigger (trigger level -40 dBm, offset is -0.2 ms)

Line 155 (79 ms): rssifs_confTrg (io, 1, -40.0, VI_FALSE)
Line 157 (40 ms): rssifs_confTrgDelay (io, -0.2e-3)
Line 160 (1424 ms): rssifs_actSendTrgWopc (io, 5000)

     Data are returned only if the trigger condition meets

Line 163 (1 ms): rssifs_readCompleteSweepData (io, traceData, &samplesReturned)
     Function Call Status: 0x3FFC09F1, WARNING: Data retrieved from trace cache.
     Instrument Error: 0x0, No error.

Line 167 (0 ms): rssifs_close (io)
```

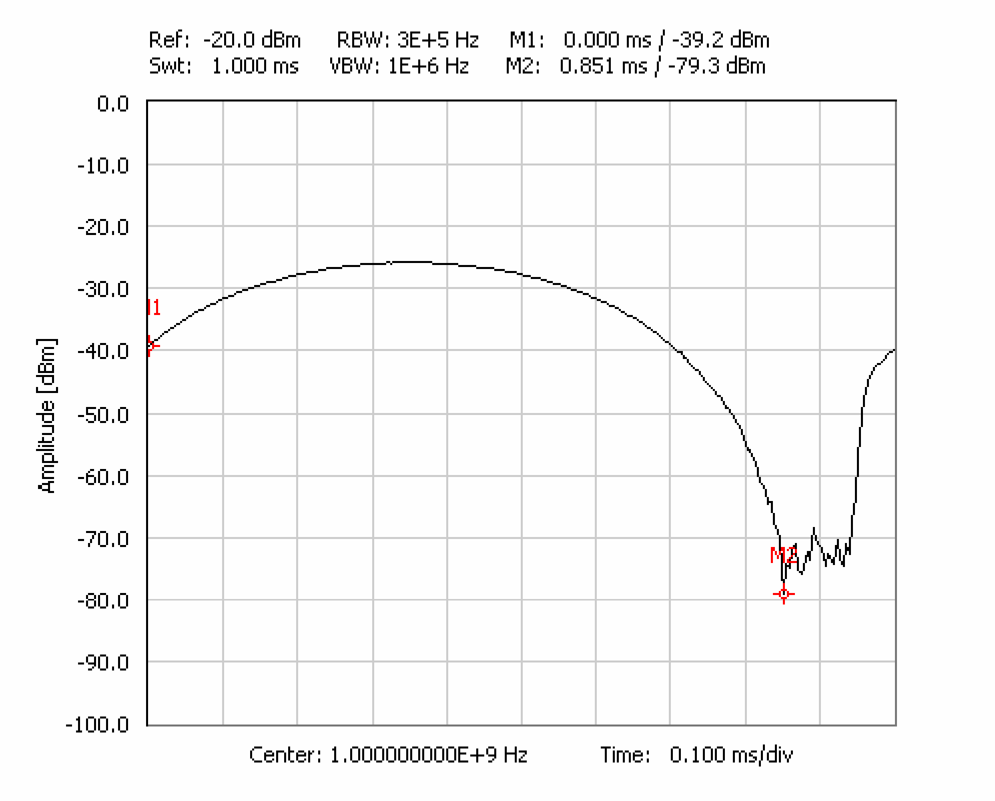
1.11.4.3 Display Results



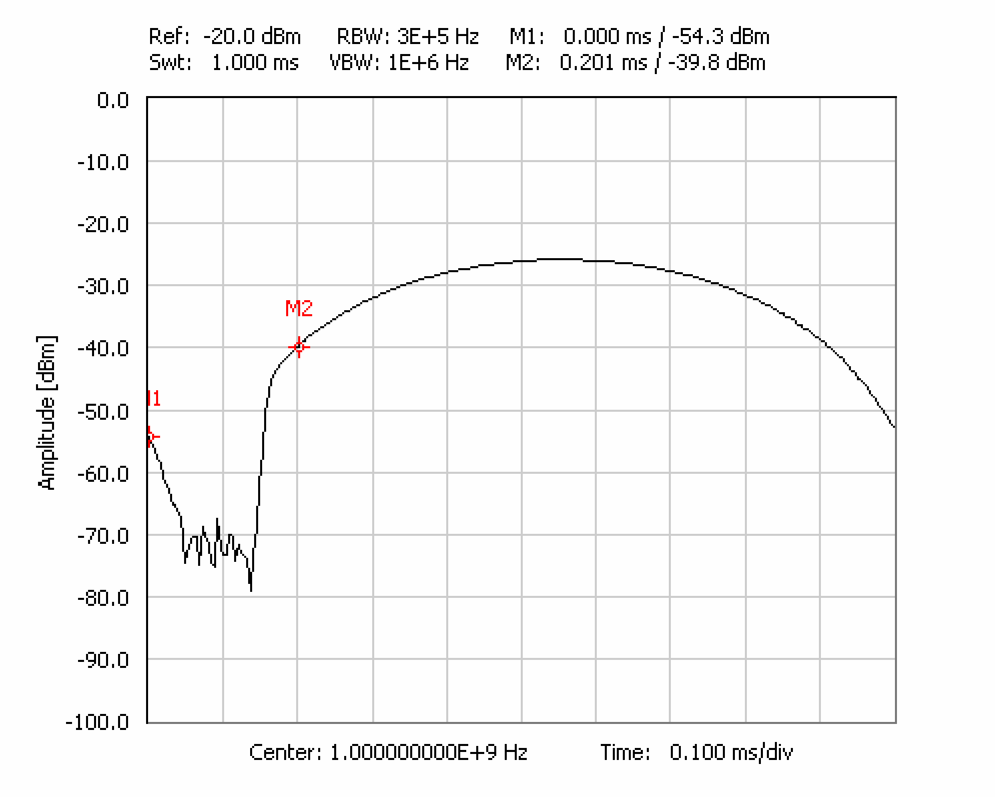
Single Sweep Mode (trace data illustration)



Triggering in the free run mode



Video Trigger (trigger level -40 dBm, trigger offset is zero)



Video Trigger (trigger level -40 dBm, offset is -0.2 ms)

1.11.5 Marker Measurement(s)

1.11.5.1 Source Code

```
/*****************************************************************************
 *
 * Title:   Marker(s) Measurement
 *
 * Purpose: This example shows how to proceed with marker measurement.
 *
 *****************************************************************************/

#include <ansi_c.h>
#include "rssifs.h"

/*** Macros & definitions ***************************************************/

#define CHECKERR( fCal ) \
    { \
        fCalStartTime = clock();\
        error = (fCal);\
        status = error;\
        printf(" Line %ld (%ld ms): %s\n", __LINE__,\
            (fCalTime = (clock() - fCalStartTime)), #fCal);\
        if (error != VI_SUCCESS)\
            {\
            rssifs_error_message (io, error, error_message);\
            printf("\tFunction Call Status: 0x%08X, %s\n", error, error_message);\
            rssifs_error_query(io, &error, error_message);\
            printf("\tInstrument Error: 0x%X, %s\n", error, error_message);\
            }\
    }

#define RESOURCE_NAME   "USB::0xAAD::0x6::100202"   // Resource name

/*** Main *******************************************************************/

int main (int argc, char *argv[])
{
    ViStatus    error               = VI_SUCCESS,
                status              = VI_SUCCESS;
    clock_t     fCalTime            = 0,
                fCalStartTime       = 0;
    ViChar      error_message[256];
    ViSession   io;

    /* Define remote connection parameters (as default values) */

    ViBoolean   IDQuery             = VI_TRUE,
                resetDevice         = VI_TRUE;
    ViRsrc      resourceName        = RESOURCE_NAME;

    ViReal64    marker_position     = 0.0,
                marker_value        = 0.0;

    CHECKERR (rssifs_init (resourceName, IDQuery, resetDevice, &io));

    printf (

    "\n --- Marker(s) Measurement ----------------------------------------------\n\n"
    "\tRF Generator is connected to the analyzer's input\n"
    "\t- Frequency: 1 GHz\n"
    "\t- Level: -30.0 dBm\n"
    "\t- Modulation: AM, modulation frequency 50 kHz, modulation depth 100 %%\n"
    "\n ------------------------------------------------------------------------\n\n"

    );

    printf ("\n --- Set center 1 GHz and span 0.5 MHz\n\n");
    CHECKERR (rssifs_confSpanCenterFrq (io, 0, 500.0e3, 1.0e9));

    /* For higher resolution is set 500 measured points per sweep */
    CHECKERR (rssifs_confSweepPoints (io, 500));
    /* Configure RBW & VBW */
    CHECKERR (rssifs_configureBandwidth (io, 8, 15));
    /* Set reference level */
    CHECKERR (rssifs_confRefLevel (io, -20.0));
    /* Set single sweep mode */
    CHECKERR (rssifs_confSweep (io, 1, 0));

    printf ("\n --- Send trigger and wait for operation completed\n\n");
    CHECKERR (rssifs_actSendTrgWopc (io, 20000));

    /* Enable markers */
    CHECKERR (rssifs_confMarkState (io, 1, VI_TRUE));
    CHECKERR (rssifs_confMarkState (io, 2, VI_TRUE));

    printf ("\n --- Search for max peak (marker 1 and 2)\n\n");

    CHECKERR (rssifs_actMarkSearch (io, 1, 0));
    CHECKERR (rssifs_actMarkSearch (io, 2, 0));

    CHECKERR (rssifs_getMarkerPosition (io, 1, &marker_position));
    CHECKERR (rssifs_readMarkerValue (io, 1, &marker_value));

    printf ("\n\t1st Marker position [Hz]: %.1Lf\n", marker_position);
    printf ("\t1st Marker Value [Vrms]: %Le\n", marker_value);

    CHECKERR (rssifs_getMarkerPosition (io, 2, &marker_position));
    CHECKERR (rssifs_readMarkerValue (io, 2, &marker_value));

    printf ("\n\t2nd Marker position [Hz]: %.1Lf\n", marker_position);
    printf ("\t2nd Marker Value [Vrms]: %Le\n", marker_value);

    printf ("\n --- Search for the left maximum (marker 1, absolute search mode)\n\n");

    CHECKERR (rssifs_actMarkSearch (io, 1, 2));

    CHECKERR (rssifs_getMarkerPosition (io, 1, &marker_position));
    CHECKERR (rssifs_readMarkerValue (io, 1, &marker_value));

    printf ("\n\t1st Marker position [Hz]: %.1Lf\n", marker_position);
    printf ("\t1st Marker Value [Vrms]: %Le\n", marker_value);

    printf ("\n --- Search for the right maximum (marker 2, absolute search mode)\n\n");

    CHECKERR (rssifs_actMarkSearch (io, 2, 3));

    CHECKERR (rssifs_getMarkerPosition (io, 2, &marker_position));
    CHECKERR (rssifs_readMarkerValue (io, 2, &marker_value));

    printf ("\n\t2nd Marker position [Hz]: %.1Lf\n", marker_position);
    printf ("\t2nd Marker Value [Vrms]: %Le\n", marker_value);
```

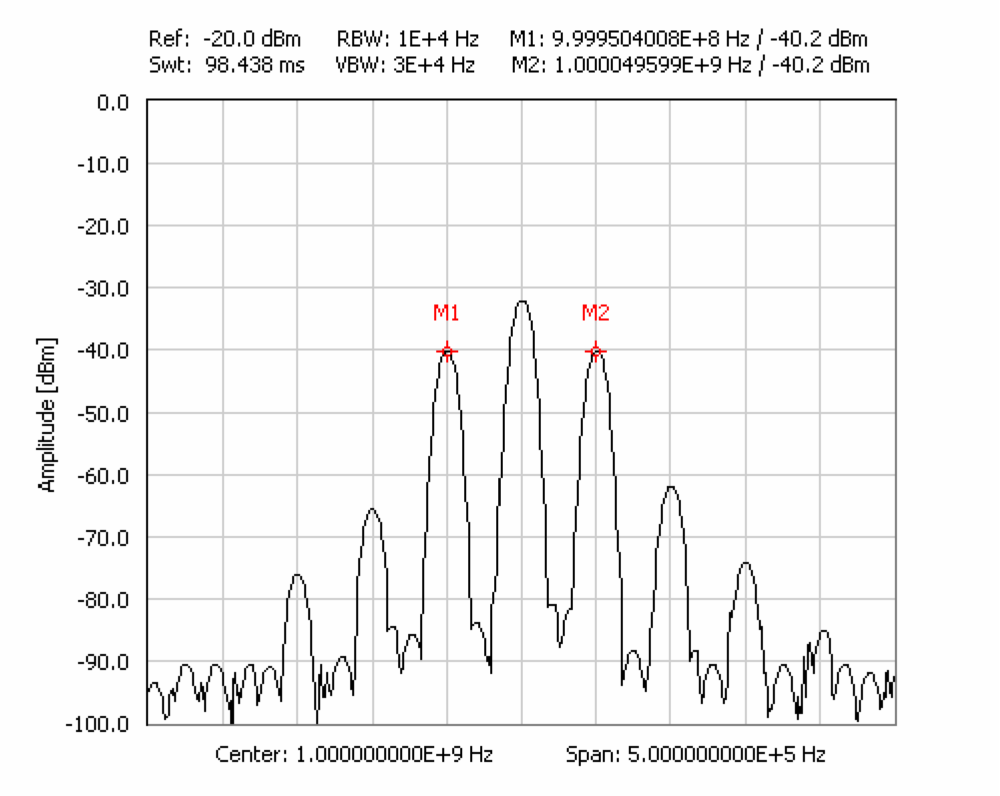
```
    CHECKERR (rssifs_close (io));

    return 0;
}
```

1.11.5.2 Execution Result

```
Line 52 (1413 ms): rssifs_init (resourceName, IDQuery, resetDevice, &io)

 --- Marker(s) Measurement -----------------------------------------------

      RF Generator is connected to the analyzer's input
      - Frequency: 1 GHz
      - Level: -30.0 dBm
      - Modulation: AM, modulation frequency 50 kHz, modulation depth 100 %

 -------------------------------------------------------------------------

 --- Set center 1 GHz and span 0.5 MHz

 Line 66 (155 ms): rssifs_confSpanCenterFrq (io, 0, 500.0e3, 1.0e9)
 Line 69 (4737 ms): rssifs_confSweepPoints (io, 500)
 Line 71 (62 ms): rssifs_configureBandwidth (io, 8, 15)
 Line 73 (39 ms): rssifs_confRefLevel (io, -20.0)
 Line 75 (367 ms): rssifs_confSweep (io, 1, 0)

 --- Send trigger and wait for operation completed

 Line 78 (231 ms): rssifs_actSendTrgWopc (io, 20000)
 Line 81 (0 ms): rssifs_confMarkState (io, 1, VI_TRUE)
 Line 82 (0 ms): rssifs_confMarkState (io, 2, VI_TRUE)

 --- Search for max peak (marker 1 and 2)

 Line 86 (3 ms): rssifs_actMarkSearch (io, 1, 0)
 Line 87 (0 ms): rssifs_actMarkSearch (io, 2, 0)
 Line 89 (0 ms): rssifs_getMarkerPosition (io, 1, &marker_position)
 Line 90 (4 ms): rssifs_readMarkerValue (io, 1, &marker_value)

      1st Marker position [Hz]: 1000000000.0
      1st Marker Value [Vrms]: 5.612415e-03
 Line 95 (0 ms): rssifs_getMarkerPosition (io, 2, &marker_position)
 Line 96 (7 ms): rssifs_readMarkerValue (io, 2, &marker_value)

      2nd Marker position [Hz]: 1000000000.0
      2nd Marker Value [Vrms]: 5.612415e-03

 --- Search for the left maximum (marker 1, absolute search mode)

 Line 103 (0 ms): rssifs_actMarkSearch (io, 1, 2)
 Line 105 (0 ms): rssifs_getMarkerPosition (io, 1, &marker_position)
 Line 106 (7 ms): rssifs_readMarkerValue (io, 1, &marker_value)

      1st Marker position [Hz]: 999950000.0
      1st Marker Value [Vrms]: 2.163790e-03

 --- Search for the right maximum (marker 2, absolute search mode)

 Line 113 (0 ms): rssifs_actMarkSearch (io, 2, 3)
 Line 115 (0 ms): rssifs_getMarkerPosition (io, 2, &marker_position)
 Line 116 (7 ms): rssifs_readMarkerValue (io, 2, &marker_value)
```

```
     2nd Marker position [Hz]: 1000050000.0
     2nd Marker Value [Vrms]: 2.181321e-03
Line 121 (0 ms): rssifs_close (io)
```

1.11.5.3 Display Results



Marker(s) Measurement

1.11.6 Marker Counter Measurement

1.11.6.1 Source Code

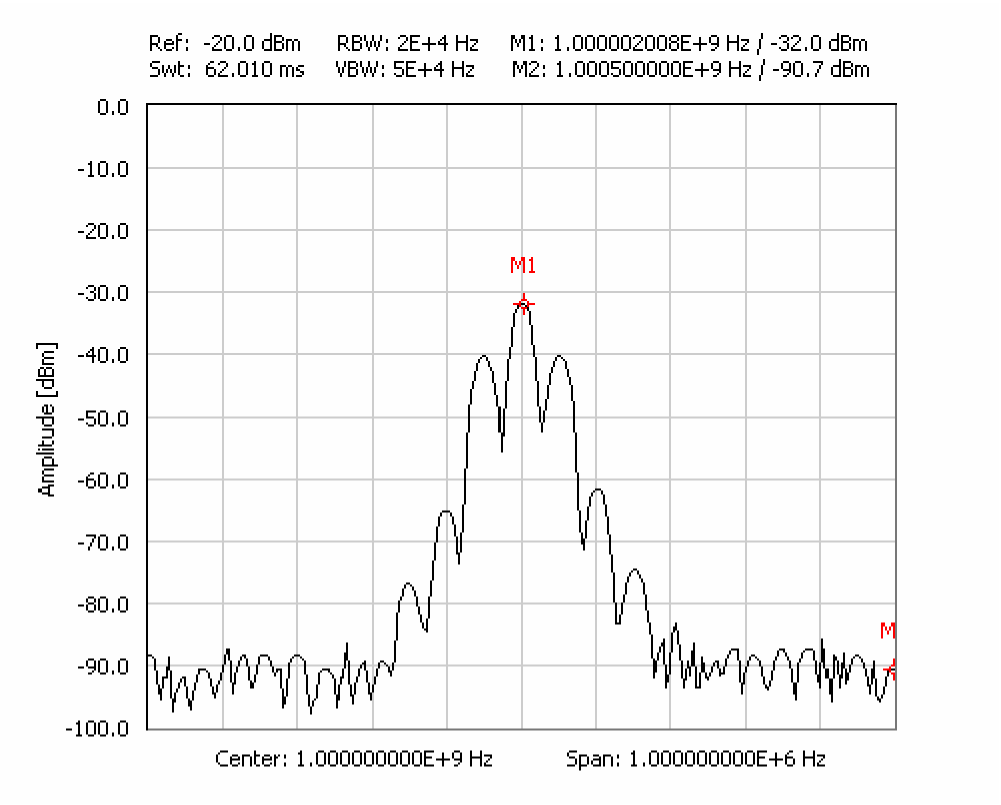
```
/*****************************************************************************
 *
 * Title:   Marker Counter Measurement
 *
 * Purpose: This example shows how to proceed with marker counter measurement.
 *
 *****************************************************************************/

#include <ansi_c.h>
#include "rssifs.h"

/*** Macros & definitions ****************************************************/

#define CHECKERR( fCal ) \
    { \
        fCalStartTime = clock();\
        error = (fCal);\
        status = error;\
        printf(" Line %ld (%ld ms): %s\n", __LINE__,\
            (fCalTime = (clock() - fCalStartTime)), #fCal);\
        if (error != VI_SUCCESS)\
            {\
            rssifs_error_message (io, error, error_message);\
            printf("\tFunction Call Status: 0x%08X, %s\n", error, error_message);\
            rssifs_error_query(io, &error, error_message);\
            printf("\tInstrument Error: 0x%X, %s\n", error, error_message);\
            }\
    }

#define RESOURCE_NAME   "USB::0xAAD::0x6::100202"   // Resource name

/*** Main ********************************************************************/

int main (int argc, char *argv[])
{
    ViStatus    error               = VI_SUCCESS,
                status              = VI_SUCCESS;
    clock_t     fCalTime            = 0,
                fCalStartTime       = 0;
    ViChar      error_message[256];
    ViSession   io;

    /* Define remote connection parameters (as default values) */

    ViBoolean   IDQuery             = VI_TRUE,
                resetDevice         = VI_TRUE;
    ViRsrc      resourceName        = RESOURCE_NAME;

    ViReal64    freq_counter        = 0.0;

    CHECKERR (rssifs_init (resourceName, IDQuery, resetDevice, &io));

    printf (

    "\n --- Marker Counter Measurement ------------------------------------------\n\n"
    "\tRF Generator is connected to the analyzer's input\n"
    "\t- Frequency: 1 GHz\n"
    "\t- Level: -30.0 dBm\n"
    "\t- Modulation: AM, modulation frequency 50 kHz, modulation depth 100 %%\n"
    "\n ------------------------------------------------------------------------\n\n"

    );

    printf ("\n --- Set center 1 GHz and span 1.0 MHz\n\n");
    CHECKERR (rssifs_confSpanCenterFrq (io, 0, 1.0e6, 1.0e9));

    /* Set reference level */
    CHECKERR (rssifs_confRefLevel (io, -20.0));
    /* Update data in the trace cache */
    CHECKERR (rssifs_actSendTrgWopc (io, 10000));

    /* Enable marker */
    CHECKERR (rssifs_confMarkState (io, 1, VI_TRUE));

    printf ("\n --- Set marker to 1 GHz\n\n");

    CHECKERR (rssifs_confMarkPosition (io, 1, 1.0e9));

    printf ("\n --- Configure and activate marker counter\n\n");

    CHECKERR (rssifs_confMarkFreqCnt (io, 1, 1, VI_TRUE));

    CHECKERR (rssifs_readMarkerCounterValue (io, 1, &freq_counter));

    printf ("\n\tFrequency Counter Result [Hz]: %.1Lf\n\n", freq_counter);

    CHECKERR (rssifs_close (io));

    return 0;
}
```

1.11.6.2 Execution Result

```
Line 51 (1442 ms): rssifs_init (resourceName, IDQuery, resetDevice, &io)

 --- Marker Counter Measurement ----------------------------------------

       RF Generator is connected to the analyzer's input
       - Frequency: 1 GHz
       - Level: -30.0 dBm
       - Modulation: AM, modulation frequency 50 kHz, modulation depth 100 %

 ------------------------------------------------------------------------


 --- Set center 1 GHz and span 1.0 MHz

 Line 65 (149 ms): rssifs_confSpanCenterFrq (io, 0, 1.0e6, 1.0e9)
 Line 68 (39 ms): rssifs_confRefLevel (io, -20.0)
 Line 70 (607 ms): rssifs_actSendTrgWopc (io, 10000)
 Line 73 (0 ms): rssifs_confMarkState (io, 1, VI_TRUE)

 --- Set marker to 1 GHz

 Line 77 (0 ms): rssifs_confMarkPosition (io, 1, 1.0e9)

 --- Configure and activate marker counter
```

```
Line 81 (54 ms): rssifs_confMarkFreqCnt (io, 1, 1, VI_TRUE)
Line 83 (1408 ms): rssifs_readMarkerCounterValue (io, 1, &freq_counter)

     Frequency Counter Result [Hz]: 1000000107.2

Line 87 (1 ms): rssifs_close (io)
```

1.11.6.3 Display Result



Marker Counter Measurement

1.11.7 Evaluation of Trace Data

1.11.7.1 Source Code

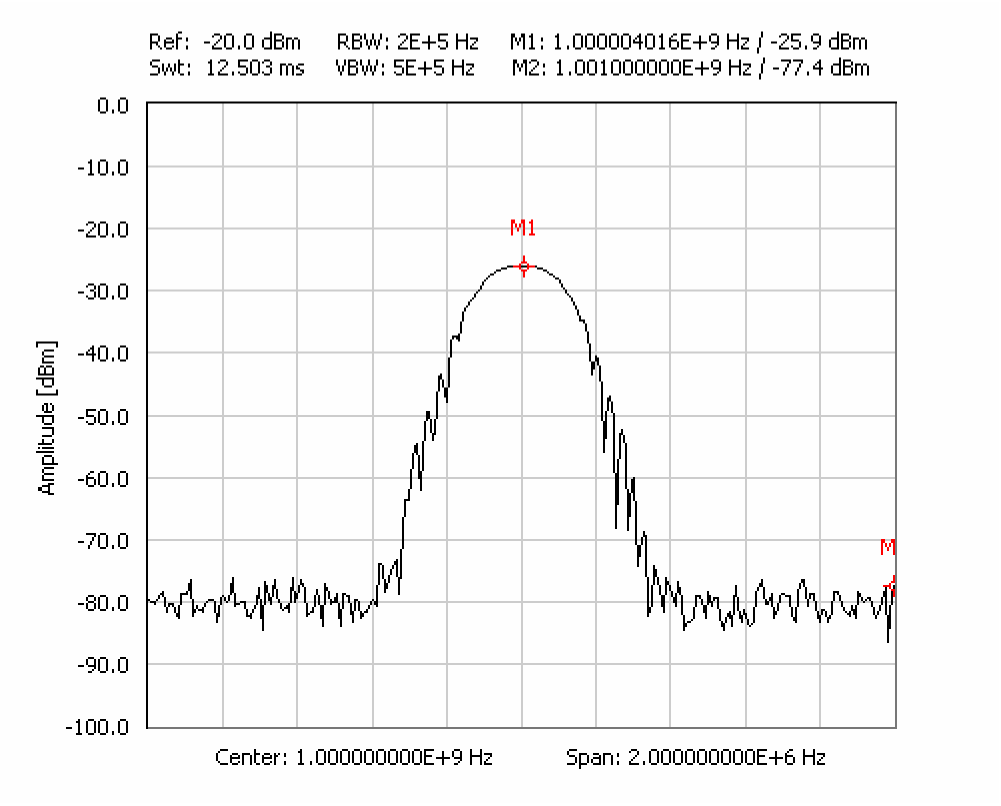
```
/*****************************************************************************
 *
 * Title:   Evaluation of Trace Data
 *
 * Purpose: This example shows how to handle trace data.
 *
 *****************************************************************************/

#include <ansi_c.h>
#include "rssifs.h"

/*** Macros & definitions ***************************************************/

#define CHECKERR( fCal ) \
    { \
        fCalStartTime = clock();\
        error = (fCal);\
        status = error;\
        printf(" Line %ld (%ld ms): %s\n", __LINE__,\
            (fCalTime = (clock() - fCalStartTime)), #fCal);\
        if (error != VI_SUCCESS)\
            {\
            rssifs_error_message (io, error, error_message);\
            printf("\tFunction Call Status: 0x%08X, %s\n", error, error_message);\
            rssifs_error_query(io, &error, error_message);\
            printf("\tInstrument Error: 0x%X, %s\n", error, error_message);\
            }\
    }

#define RESOURCE_NAME   "USB::0xAAD::0x6::100202"   // Resource name

/*** Main *******************************************************************/

int main (int argc, char *argv[])
{
    ViStatus    error               = VI_SUCCESS,
                status              = VI_SUCCESS;
    clock_t     fCalTime            = 0,
                fCalStartTime       = 0;
    ViChar      error_message[256];
    ViSession   io;

    ViReal64    sweep_time          = 0.0,
                traceData[2048];

    ViInt32     samplesReturned     = 0;

    /* Define remote connection parameters (as default values) */

    ViBoolean   IDQuery             = VI_TRUE,
                resetDevice         = VI_TRUE;
    ViRsrc      resourceName        = RESOURCE_NAME;

    CHECKERR (rssifs_init (resourceName, IDQuery, resetDevice, &io));

    printf (
    "\n --- Evaluation of Trace Data -------------------------------------------\n\n"
    "\tRF Generator is connected to the analyzer's input\n"
    "\t- Frequency: 1 GHz\n"
    "\t- Level: -30.0 dBm\n"
    "\t- Modulation: AM, modulation frequency 5 kHz, modulation depth 100 %%\n"
    "\n ------------------------------------------------------------------------\n\n"

    );

    /* Set reference level */
    CHECKERR (rssifs_confRefLevel (io, -20.0));
    /* Set RBW to 200 kHz VBW to 500 kHz */
    CHECKERR (rssifs_configureBandwidth (io, 13, 20));
    /* Set start and stop frequency */
    CHECKERR (rssifs_confStartStopFrq (io, 0, 999.0e6, 001e9));
    /* Get Sweep Time */
    CHECKERR (rssifs_getSweepTime (io, &sweep_time));
    printf ("\tSweep time is adjusted to %.3Lf s\n", sweep_time);

    printf ("\n --- Read complete sweep (Clear/Write mode)\n\n");

    /* Configure Trace Mode Clear/Write */
    CHECKERR (rssifs_confTraceMode (io, 0));
    /* Read trace data */
    CHECKERR (rssifs_readCompleteSweepData (io, traceData, &samplesReturned));

    printf ("\n --- Read complete sweep (Average mode)\n\n");

    /* Configure single sweep mode to 10 sweeps per trigger */
    CHECKERR (rssifs_confSweep (io, 2, 10));
    /* Configure Trace Mode to Average */
    CHECKERR (rssifs_confTraceMode (io, 2));
    /* Start measurement and read the trace data */
    CHECKERR (rssifs_actSendTrgWopc (io, 5000));
    CHECKERR (rssifs_readCompleteSweepData (io, traceData, &samplesReturned));

    printf (

    "\n\tWarning code informs you that sweep data are after trigger\n"
    "\tis executed stored to the driver cache and function\n"
    "\trssifs_readCompleteSweepData is not reading data from instrument.\n"

    );

    printf ("\n --- Read raw trace data (bypassing trace caching mechanism)\n\n");

    {
        ViReal64    data[2048],
                    tempData[2048];
        ViInt32     startIndex      = 0,
                    samplesCount    = 0,
                    sweepPoints     = 500;

        /* Configure start and stop frequency */
        CHECKERR( rssifs_confStartStopFrq (io, 0, 500.0e6, 1.5e9));
        /* Set RBW to 100 kHz VBW to 300 kHz */
        CHECKERR (rssifs_configureBandwidth (io, 12, 19));
        /* Configure number of sweep points */
        CHECKERR( rssifs_confSweepPoints (io, sweepPoints));
        /* Configure continuous sweep mode */
        CHECKERR( rssifs_confSweep (io, 0, 0));
        /* Configure Sweep Time */
        CHECKERR (rssifs_confSweepTime (io, 1.0));
        CHECKERR (rssifs_getSweepTime (io, &sweep_time));
        printf ("\tSweep time is adjusted to %.3Lf s\n", sweep_time);

        printf ("\n ... Restart measurement and wait for result\n\n");
        CHECKERR( rssifs_actSendTrgWopc (io, 5000));

        printf ("\n ... Do once dummy read of trace data (flush instr buffer)\n\n");

        CHECKERR( rssifs_readTraceData (io, data, &startIndex, &samplesCount));
        printf ("\tStart index = \"%ld\", samples returned = \"%ld\"\n",
            startIndex, samplesCount);

        /* Delay for cca 1300 millisecond (illustrative example) */
        {
            clock_t start_time = clock();

            while ((long)(clock() - start_time) < 1300);
        }

        /* --- Simplified method of reading raw trace data --- */

        printf ("\n --- Simplified method of reading raw trace data\n\n");

        CHECKERR (rssifs_readTraceData (io, data, &startIndex, &samplesCount));
        printf ("\tStart index = \"%ld\", samples returned = \"%ld\"\n\n",
            startIndex, samplesCount);

            {
            ViUInt16    count       = samplesCount,
                        originStart = startIndex;

             while ((count < sweepPoints) && (count < 2048))
                {

                CHECKERR (rssifs_readTraceData (io, &tempData[count],
                    &startIndex, &samplesCount));
                printf ("\tStart index = \"%ld\", samples returned = \"%ld\"\n\n",
                    startIndex, samplesCount);

                if (samplesCount == sweepPoints) /* Full sweep? */
                    {
                    originStart = startIndex;
                    break;
                    }

                if (samplesCount == 0) /* Empty buffer? */
                    {
                    printf ("\n\t*** Data not available ***\n\n");
                    break;
                    }

                count += samplesCount;
                }

            /* Samples count should be then equal to the number of sweep points */
            samplesCount = sweepPoints;

            /* Place data to the target buffer in proper order */
            memcpy (&traceData[originStart], tempData,
                (sweepPoints - originStart) * sizeof (tempData[0]));
            memcpy (traceData, &tempData[(sweepPoints - originStart)],
                originStart * sizeof (tempData[0]));
            }
    }

    CHECKERR (rssifs_close (io));

    return 0;
}
```

1.11.7.2 Execution Result

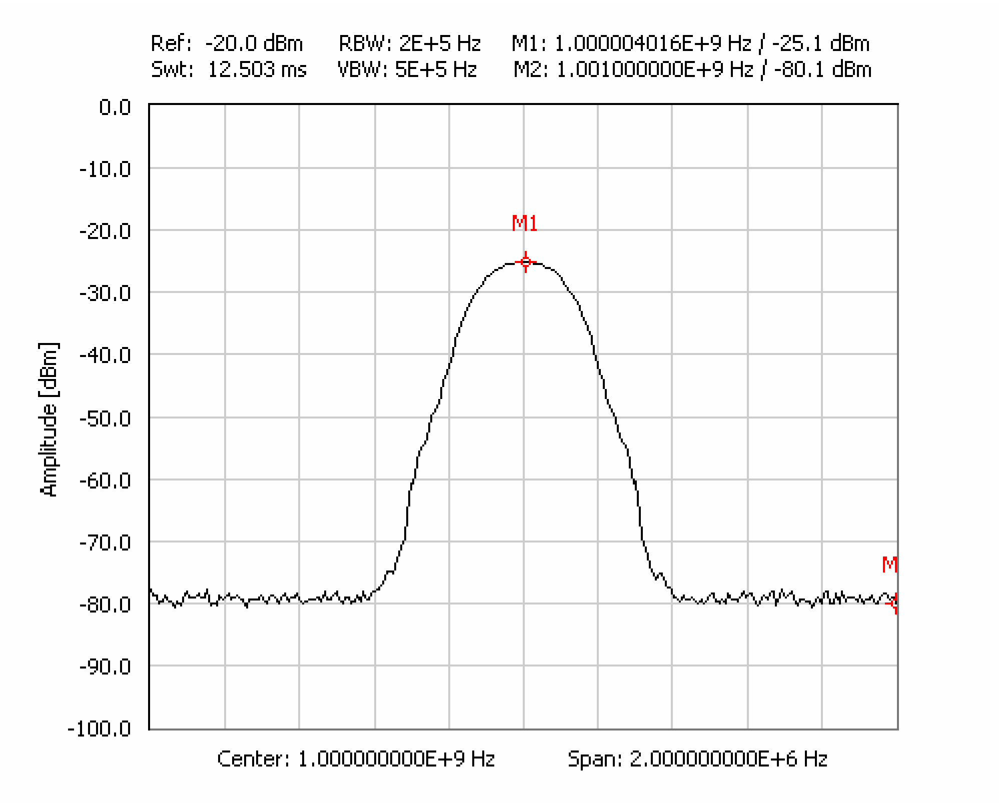
```
Line 54 (1498 ms): rssifs_init (resourceName, IDQuery, resetDevice, &io)

 --- Evaluation of Trace Data -------------------------------------------

      RF Generator is connected to the analyzer's input
      - Frequency: 1 GHz
      - Level: -30.0 dBm
      - Modulation: AM, modulation frequency 5 kHz, modulation depth 100 %

 ------------------------------------------------------------------------

 Line 68 (37 ms): rssifs_confRefLevel (io, -20.0)
 Line 70 (64 ms): rssifs_configureBandwidth (io, 13, 20)
 Line 72 (64 ms): rssifs_confStartStopFrq (io, 0, 999.0e6, 001e9)
 Line 74 (24 ms): rssifs_getSweepTime (io, &sweep_time)
      Sweep time is adjusted to 1.240 s

 --- Read complete sweep (Clear/Write mode)

 Line 80 (1 ms): rssifs_confTraceMode (io, 0)
 Line 82 (127 ms): rssifs_readCompleteSweepData (io, traceData, &samplesReturned)

 --- Read complete sweep (Average mode)

 Line 87 (312 ms): rssifs_confSweep (io, 2, 10)
 Line 89 (0 ms): rssifs_confTraceMode (io, 2)
 Line 91 (1432 ms): rssifs_actSendTrgWopc (io, 5000)
 Line 92 (0 ms): rssifs_readCompleteSweepData (io, traceData, &samplesReturned)
      Function Call Status: 0x3FFC09F1, WARNING: Data retrieved from trace cache.
      Instrument Error: 0x0, No error.

      Warning code informs you that sweep data are after trigger
      is executed stored to the driver cache and function
      rssifs_readCompleteSweepData is not reading data from instrument.

 --- Read raw trace data (bypassing trace caching mechanism)

 Line 112 (63 ms): rssifs_confStartStopFrq (io, 0, 500.0e6, 1.5e9)
 Line 114 (56 ms): rssifs_configureBandwidth (io, 12, 19)
 Line 116 (4143 ms): rssifs_confSweepPoints (io, sweepPoints)
 Line 118 (214 ms): rssifs_confSweep (io, 0, 0)
 Line 120 (158 ms): rssifs_confSweepTime (io, 1.0)
 Line 121 (24 ms): rssifs_getSweepTime (io, &sweep_time)
      Sweep time is adjusted to 1.000 s

 ... Restart measurement and wait for result

 Line 125 (1656 ms): rssifs_actSendTrgWopc (io, 5000)

 ... Do once dummy read of trace data (flush instrument buffer)

 Line 129 (24 ms): rssifs_readTraceData (io, data, &startIndex, &samplesCount)
```

```
      Start index = "0", samples returned = "500"

 --- Simplified method of reading raw trace data

 Line 144 (36 ms): rssifs_readTraceData (io, data, &startIndex, &samplesCount)
      Start index = "0", samples returned = "10"

 Line 156 (23 ms): rssifs_readTraceData (io, &tempData[count], &startIndex,
&samplesCount)
      Start index = "10", samples returned = "490"

 Line 186 (0 ms): rssifs_close (io)
```

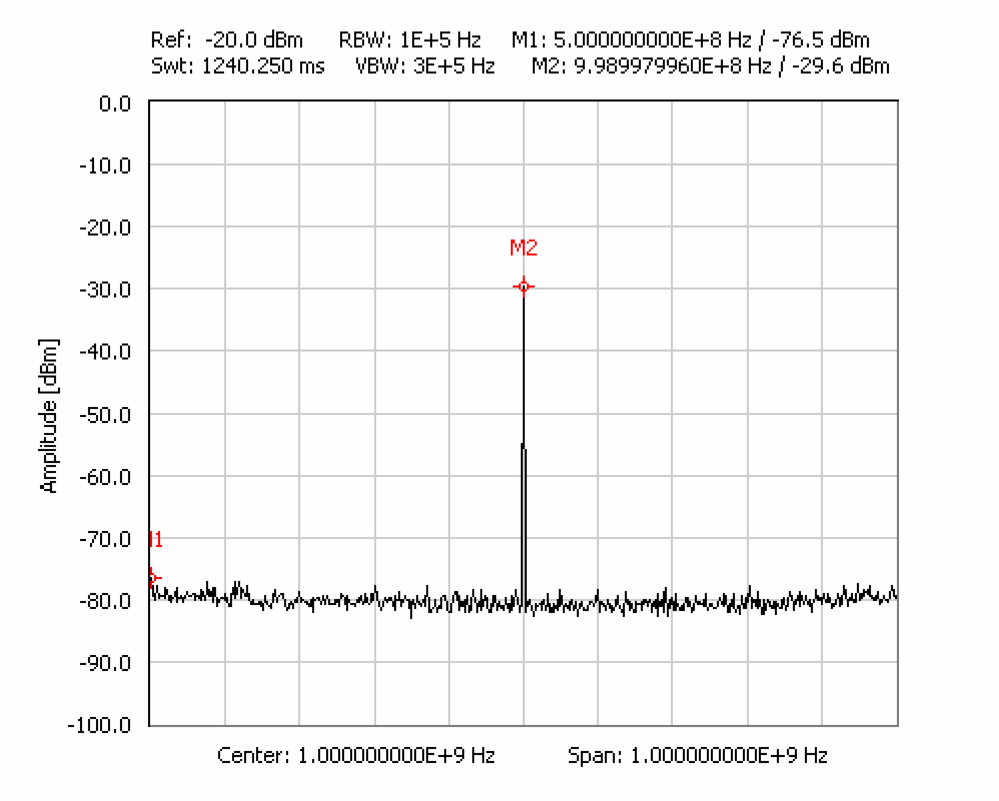
1.11.7.3 Display Result



Complete sweep (Clear/Write mode)



Complete sweep (Averaged over 10 sweeps)



Result of reading raw trace data

1.11.8 Sweep Time & Resolution Bandwidth & Video Bandwidth

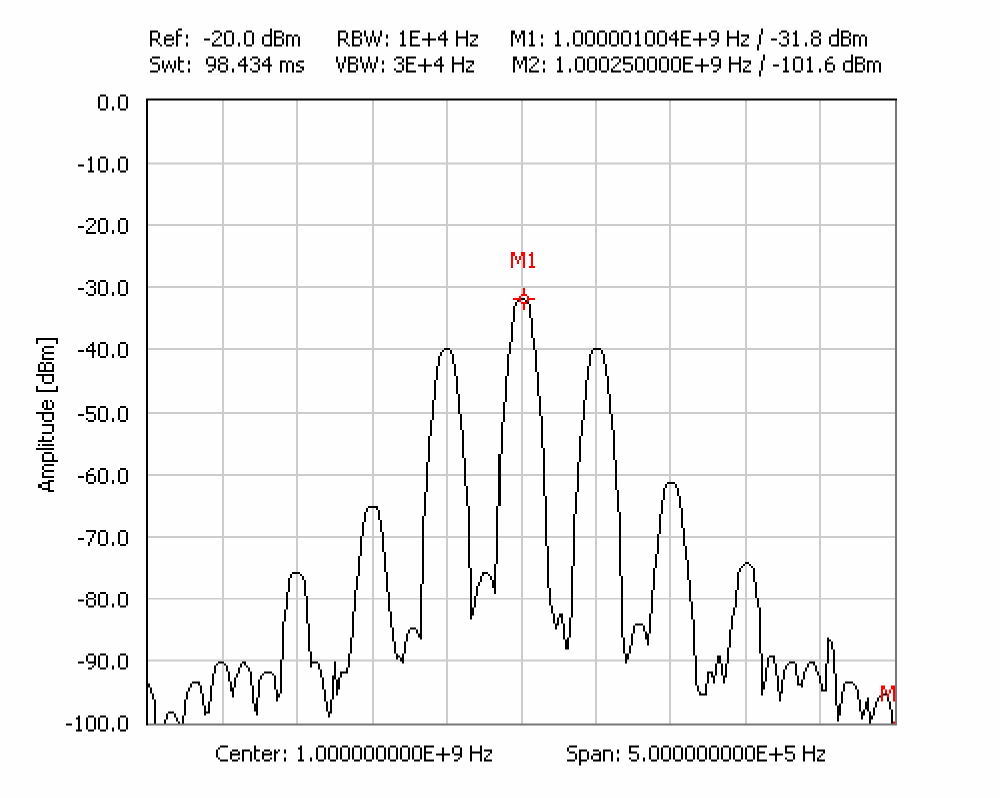
1.11.8.1 Source Code

```
/*****************************************************************************
 *
 * Title:   Sweep Time & Resolution Bandwidth & Video Bandwidth
 *
 * Purpose: This example shows dependencies of sweep time, resolution
 *          bandwidth and video bandwidth.
 *
 *****************************************************************************/

#include <ansi_c.h>
#include "rssifs.h"

/*** Macros & definitions ****************************************************/

#define CHECKERR( fCal ) \
    { \
        fCalStartTime = clock();\
        error = (fCal);\
        status = error;\
        printf(" Line %ld (%ld ms): %s\n", __LINE__,\
            (fCalTime = (clock() - fCalStartTime)), #fCal);\
        if (error != VI_SUCCESS)\
            {\
            rssifs_error_message (io, error, error_message);\
            printf("\tFunction Call Status: 0x%08X, %s\n", error, error_message);\
            rssifs_error_query(io, &error, error_message);\
            printf("\tInstrument Error: 0x%X, %s\n", error, error_message);\
            }\
    }

#define RESOURCE_NAME   "USB::0xAAD::0x6::100202"   // Resource name

/*** Main ********************************************************************/

int main (int argc, char *argv[])
{
    ViStatus    error               = VI_SUCCESS,
                status              = VI_SUCCESS;
    clock_t     fCalTime            = 0,
                fCalStartTime       = 0;
    ViChar      error_message[256];
    ViSession   io;

    /* Define remote connection parameters (as default values) */

    ViBoolean   IDQuery             = VI_TRUE,
                resetDevice         = VI_TRUE;
    ViRsrc      resourceName        = RESOURCE_NAME;

    ViReal64    sweep_time          = 0.0,
                RBW                 = 0.0,
                VBW                 = 0.0;

    CHECKERR (rssifs_init (resourceName, IDQuery, resetDevice, &io));

    printf (
    "\n --- Sweep Time & Resolution Bandwidth & Video Bandwidth ----------------\n\n"
    "\tRF Generator is connected to the analyzer's input\n"
    "\t- Frequency: 1 GHz\n"
    "\t- Level: -30.0 dBm\n"
    "\t- Modulation: AM, modulation frequency 50 kHz, modulation depth 100 %%\n"
    "\n ------------------------------------------------------------------------\n\n"

    );

    /* Set reference level */
    CHECKERR (rssifs_confRefLevel (io, -20.0));
    /* Set single sweep mode */
    CHECKERR (rssifs_confSweep (io, 1, 0));

    printf ("\n --- Auto adjusting of RBW, VBW and Sweep Time\n\n");

    CHECKERR (rssifs_confSweepTime (io, 0.0));
    CHECKERR (rssifs_configureBandwidth (io, 0, 0));

    printf ("\n --- Set center 1 GHz and span 500 kHz\n\n");

    CHECKERR (rssifs_confSpanCenterFrq (io, 0, 500.0e3, 1.0e9));
    CHECKERR (rssifs_actSendTrg (io));

    printf ("\nGet settings:\n\n");

    CHECKERR (rssifs_getSweepTime (io, &sweep_time));
    CHECKERR (rssifs_getResolutionBandwidth (io, &RBW));
    CHECKERR (rssifs_getVideoBandwidth (io, &VBW));

    printf ("\n\tSweep time [s]: %.3Lf\n", sweep_time);
    printf ("\tRBW [Hz]: %.3Lf\n", RBW);
    printf ("\tVBW [Hz]: %.3Lf\n", VBW);

    printf ("\n --- Set RBW to 100 kHz, VBW is auto\n\n");

    CHECKERR (rssifs_confResBW (io, 100.0e3));
    CHECKERR (rssifs_actSendTrg (io));

    printf ("\nGet settings:\n\n");

    CHECKERR (rssifs_getSweepTime (io, &sweep_time));
    CHECKERR (rssifs_getResolutionBandwidth (io, &RBW));
    CHECKERR (rssifs_getVideoBandwidth (io, &VBW));

    printf ("\n\tSweep time [s]: %.3Lf\n", sweep_time);
    printf ("\tRBW [Hz]: %.3Lf\n", RBW);
    printf ("\tVBW [Hz]: %.3Lf\n\n", VBW);

    CHECKERR (rssifs_close (io));

    return 0;
}
```

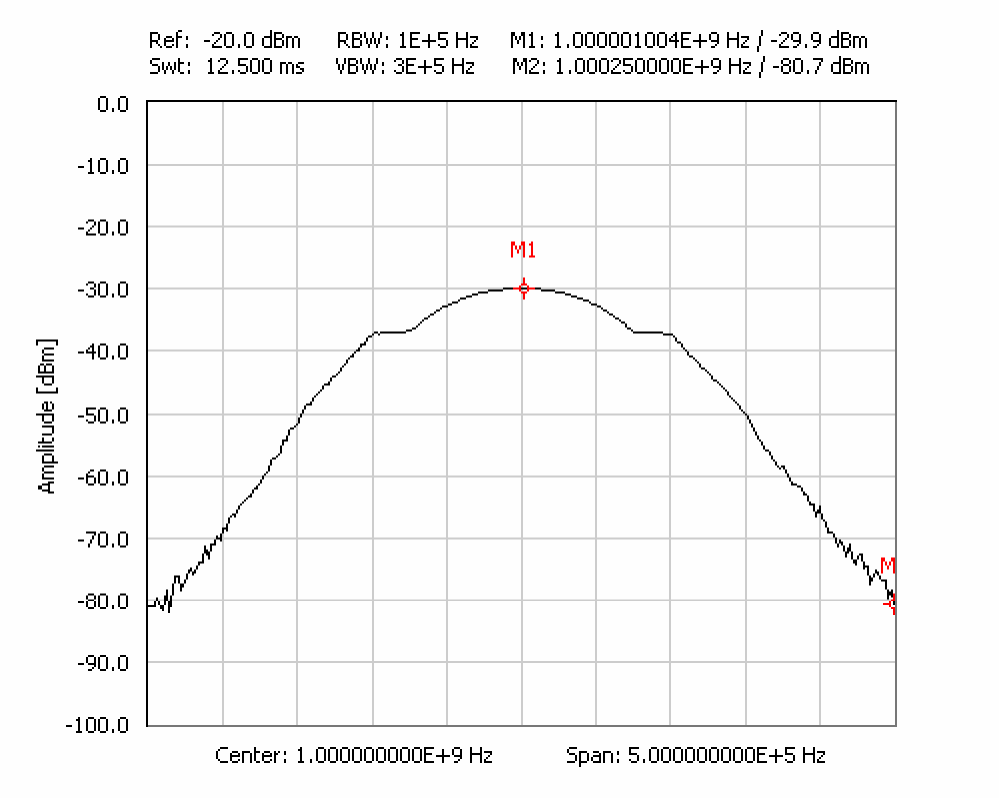
Execution Result

```
Line 54 (1489 ms): rssifs_init (resourceName, IDQuery, resetDevice, &io)

 --- Sweep Time & Resolution Bandwidth & Video Bandwidth ----------------

      RF Generator is connected to the analyzer's input
      - Frequency: 1 GHz
      - Level: -30.0 dBm
      - Modulation: AM, modulation frequency 50 kHz, modulation depth 100 %
 ------------------------------------------------------------------------

 Line 68 (37 ms): rssifs_confRefLevel (io, -20.0)
 Line 70 (640 ms): rssifs_confSweep (io, 1, 0)

 --- Auto adjusting of RBW, VBW and Sweep Time

 Line 74 (40 ms): rssifs_confSweepTime (io, 0.0)
 Line 75 (136 ms): rssifs_configureBandwidth (io, 0, 0)

 --- Set center 1 GHz and span 500 kHz

 Line 79 (168 ms): rssifs_confSpanCenterFrq (io, 0, 500.0e3, 1.0e9)
 Line 80 (40 ms): rssifs_actSendTrg (io)

Get settings:

 Line 84 (80 ms): rssifs_getSweepTime (io, &sweep_time)
 Line 85 (23 ms): rssifs_getResolutionBandwidth (io, &RBW)
 Line 86 (24 ms): rssifs_getVideoBandwidth (io, &VBW)

      Sweep time [s]: 0.098
      RBW [Hz]: 10000.000
      VBW [Hz]: 30000.000

 --- Set RBW to 100 kHz, VBW is auto

 Line 94 (95 ms): rssifs_confResBW (io, 100.0e3)
 Line 95 (43 ms): rssifs_actSendTrg (io)

Get settings:

 Line 99 (79 ms): rssifs_getSweepTime (io, &sweep_time)
 Line 100 (23 ms): rssifs_getResolutionBandwidth (io, &RBW)
 Line 101 (24 ms): rssifs_getVideoBandwidth (io, &VBW)

      Sweep time [s]: 0.013
      RBW [Hz]: 100000.000
      VBW [Hz]: 300000.000

 Line 107 (0 ms): rssifs_close (io)
```

1.11.8.2 Display Result



Measurement Result (RBW 10 kHz, VBW 30 kHz)



Measurement Result (RBW 100 kHz, VBW 300 kHz)