

# HOW TO USE ROHDE & SCHWARZ INSTRUMENTS IN MATLAB

Miloslav Macko | 1MA171 | Version 14e | 10.2021

MATLAB® is a registered trademark of the The MathWorks, Inc.

MATLAB Instrument Control Toolbox™ is a trademark of the The MathWorks, Inc.

R&S® is a registered trademark of Rohde & Schwarz® GmbH & Co. KG.

Microsoft® and Windows® are U.S. registered trademarks of the Microsoft Corporation

**Note:**

Please find the most up-to-date Application Note on our homepage:

<http://www.rohde-schwarz.com/appnote/1MA171>

**ROHDE & SCHWARZ**

Make ideas real



# Contents

<b>1</b>	<b>Overview.....</b>	<b>3</b>
<b>2</b>	<b>Introduction.....</b>	<b>4</b>
2.1	VISA Connection and Direct SCPI Commands .....	4
2.2	Using VXI plug&play Instrument Drivers .....	4
<b>3</b>	<b>Direct SCPI Commands Communication .....</b>	<b>5</b>
<b>4</b>	<b>Using VXI plug&amp;play Instrument Drivers .....</b>	<b>9</b>
4.1	Installing VXIplug&play Instrument Drivers .....	9
4.2	MATLAB MDD Drivers .....	10
4.3	Opening and Closing Instrument Driver Session.....	11
4.4	Calling Instrument Driver Functions .....	12
4.5	Setting Property Value .....	13
4.6	Reading Property Value .....	13
4.7	Repeated Capabilities .....	14
4.8	Property Identifier .....	15

# 1 Overview

This application note outlines two different approaches for remote-controlling Rohde & Schwarz instruments out of MathWorks MATLAB:

- ▶ The first one uses VISA connection and direct SCPI commands.
- ▶ The second approach takes advantage of Rohde & Schwarz VXI plug&play instrument drivers and MATLAB Instrument Control Toolbox.

## 2 Introduction

MATLAB has become widely used platform among students, engineers and developers. When users wish to remote-control measurement instruments from MATLAB, they have several options to choose from. This application note presents two of them.

### 2.1 VISA Connection and Direct SCPI Commands

We recommend this option for most of the users. It is simple and besides VISA it does not require any additional software component. Attached to this application note, is a MATLAB class **VISA\_Instrument** that presents VISA interface for MATLAB script language. Practical examples with the **VISA\_Instrument** are part of the attachment.

Advantages:

- ▶ Simplicity.
- ▶ VISA session is closed properly even if MATLAB script is interrupted; this is helpful if an instrument can only handle one VISA session at a time.
- ▶ Most of the commonly used operations are provided by the attached MATLAB class `VISA_Instrument`. This class is open for further extensions.
- ▶ Error handling is performed in the form of exceptions.
- ▶ Included, are ready-to-use examples for R&S FSW / FSV / RTO / RTE / RTB

Disadvantages:

- ▶ You need to get familiar with the instrument's SCPI language.
- ▶ Parsing of more complex instrument responses needs to be done in the user code.

### 2.2 Using VXI plug&play Instrument Drivers

This alternative route takes advantage of Rohde & Schwarz VXI plug&play instrument drivers and requires MATLAB Instrument Control Toolbox to be available.

Advantages:

- ▶ Error handling is already performed by the instrument driver.
- ▶ Instrument drivers take care of proper measurement synchronization.
- ▶ Instruments driver come with help file for all functions and attributes.
- ▶ More complex instrument responses are already parsed by instrument drivers.

Disadvantages:

- ▶ Slightly longer learning curve.
- ▶ Longer initialization of the driver session due to additional time needed to parse the MATLAB driver file.
- ▶ More difficult to build executables, since the code uses external dll libraries.

# 3 Direct SCPI Commands Communication

Referenced files - all packed into `MATLAB_directSCPI_Examples.zip`:

- ▶ `VISA_Instrument.m`
- ▶ `MATLAB_directSCPI_Hello_World.m`
- ▶ `MATLAB_directSCPI_Specan_Example.m`
- ▶ `MATLAB_directSCPI_Scope_Example.m`
- ▶ `MATLAB_directSCPI_RTB_Example.m`

Required software. The actual software used to prepare this document is mentioned in the round brackets:

- ▶ MATLAB 2013 or later (2016a)
- ▶ Windows XP / VISTA / Win 7 (Win 7 64-bit)
- ▶ NI VISA I/O library 15.0 or late (NI VISA 16.0)

If you are new to the topic of remote-control, we recommend reading this small tutorial:

[R&S Instrument Drivers and Remote Control](#)

Communication with an instrument over VISA is of a synchronous message-based type. That means, the instrument never responds unless the controller (your computer) requires it to do so. The request is formed into a string called **SCPI** command (short for **S**imple **C**ommands for **P**rogrammable **I**nstruments), and the instrument reacts in two different ways:

- ▶ processing it, but returning no response. An example of such command is `*RST` (returning your instrument to a defined state).
- ▶ processing it and returning a response. Such command contains question mark and often is more specifically called query. An example of a query is `*IDN?` (asking for the instrument identification string). After sending this query, you must read the response from the instrument.



Some SCPI commands exist both as commands and queries. An example would be Spectrum Analyzer center frequency. You can set the center frequency with the SCPI command `'FREQ:CENT 100MHz'`, and also ask for it with the query `'FREQ:CENT?'`

However, for example, the `*RST` command exists only as command, the query form `*RST?` is not valid.

In contrast, the command `*IDN?` exists only as query. The form `*IDN` is invalid.



Most common mistakes with SCPI commands and queries are:

Sending a command (not a query) and **trying to read** a response from the instrument. This results in your program waiting until the **VISA timeout** occurs, since the instrument has nothing to respond to.

Sending a query and **not reading** the response from the instrument. This causes problem with the next query, when the instrument will report the error `Query Interrupted`. It means, you did not read the previous response before you sent a new query.

To avoid both of these problems, always use the `Write()` methods for commands and `Query...()` methods for queries - see the description of the `VISA_Instrument` class below.

The `'*RST'` command and the `'*IDN?'` query are standard commands supported by every SCPI-conform instrument. To get the information about valid SCPI commands for your instrument, refer to the **Remote Control** portion of its user manual. User manual also describes whether the command is available as a command, query, or both.

To follow next steps, extract the content of the `MATLAB_directSCPI_Examples.zip` into your MATLAB working directory.

All the examples from the ZIP file show the usage of the `VISA_Instrument` object. They are written according to the description in the abovementioned remote-control tutorial, especially the chapters on **Measurement Synchronization** and **Instrument Error Checking**.

The file `VISA_Instrument.m` is a MATLAB class wrapping up .NET component called **Ivi.Visa**. It offers convenient way of communicating with your instrument and also parsing common type of responses to MATLAB-native variables. A small script below (also available as `MATLAB_directSCPI_Hello_World.m`) shows how simple is it to open a VISA instrument connection, reset the instrument, query identification string, and close the connection:

```
mySpecan = VISA_Instrument( 'TCPIP::192.168.2.100::INSTR' );  
mySpecan.Write('*RST');  
idnResponse = mySpecan.QueryString('*IDN?')  
msgbox(sprintf('Hello, I am\n%s', idnResponse));  
mySpecan.Close();
```

Most commonly used operations with examples are shown in the table below. To invoke the full help of `VISA_Instrument`, type the following into your MATLAB Command Window (without sharp brackets):

```
>> help VISA_Instrument
```

and use the provided link:

Reference page for `VISA_Instrument`

## Most commonly used VISA\_Instrument methods and properties

VISA Instrument() - constructor that opens the connection to the instrument.

**Example:**

```
mySpecan = VISA_Instrument('TCPIP::192.168.1.100::INSTR')
```

Close() - closes the connection to the instrument.

**Example:**

```
mySpecan.Close()
```

Write() - writes command to the instrument.

**Examples:**

```
mySpecan.Write('*RST')
mySpecan.Write('FREQUENCY:CENTER %0.1f', frequency)
```

AddLFtoWriteEnd - property for adding LINEFEED (0x0A or '\n') to the end of each sent command, since some instruments require it. Default value is false.

**Example:**

```
mySpecan.AddLFtoWriteEnd = true
mySpecan.Write('*RST') - the actual sent string is '*RST\n'
```

ReadString() reads response from the instrument and returns it as MATLAB string. Make sure that you only use it in combination with Write(), or better use the QueryString().

**Example:**

```
mySpecan.Write('*IDN?')
idnResponse = mySpecan.ReadString()
```

QueryString() combines Write() and ReadString() into one method.

**Examples:**

```
idnResponse = mySpecan.QueryString('*IDN?')
limitLineName = mySpecan.QueryString('CALC:LIM%d:NAME?', limitLine)
```

QueryLongString() - use this method instead of the QueryString(), if you expect a response longer than 4096 bytes.

**Examples:**

```
idnResponse = mySpecan.QueryLongString('*OPT?')
catalog = mySpecan.QueryLongString('DISPLAY:WINDOW%d:CATALOG?', window)
```

QueryBoolean(), QueryInteger(), QueryDouble() query the instrument and convert responses to MATLAB boolean, integer and double values.

**Examples:**

```
output = mySpecan.QueryBoolean('OUTPUT1?')
output = mySpecan.QueryBoolean('OUTPUT%d?', output)

assignedTrace = mySpecan.QueryInteger('CALC:MARK1:TRAC?')
assignedTrace = mySpecan.QueryInteger('CALC:MARK%d:TRAC?', marker)

markerAmplitude = mySpecan.QueryDouble('CALC:MARK1:Y?')
markerAmplitude = mySpecan.QueryDouble('CALC:MARK%d:Y?', marker)
```

## More advanced methods

`QueryBinaryFloatData()` queries binary-formatted data (traces or waveforms) from the instrument.

**Example:**

```
trace = mySpecan.QueryBinaryFloatData('FORM REAL,32;:TRAC? TRACE1')
```

`ReadBinaryDataToFile()` reads instrument binary response and stores it to a PC file. Use this, for example, to transfer a screenshot file from the instrument to the PC. Make sure that you only use it in combination with `Write()`.

**Example:**

```
mySpecan.Write('MMEM:DATA? 'c:\Instrument\Device_Screenshot.png')
mySpecan.ReadBinaryDataToFile('c:\PC_Screenshot.png')
```

`QueryASCII_ListOfDoubles()` queries comma-separated list of numbers and returns them as double array. You must define the maximum expected array size.

**Example:**

```
trace = mySpecan.QueryASCII_ListOfDoubles('FORM ASC;:TRAC? TRACE1', 100000)
```

`ErrorChecking()` throws an exception if the instrument reports an error. The procedure for checking instrument error is described in the remote-control tutorial mentioned above, chapter Instrument Error Checking. If you only want to check for instrument errors without throwing an exception, call the `ReadErrorQueue()`

**Example:**

```
mySpecan.ErrorChecking()
```

Open the attached example file for the R&S FSW / FSV / FPS:

`MATLAB_directSCPI_Specan_Example.m`

The example is commented in detail to show proper initialization, settings, acquisition of a trace, retrieving trace results, marker results and a screenshot file. Included, is proper instrument error checking and measurement synchronization.



Always use **single** acquisition mode with Spectrum Analyzers, Network Analyzers, Oscilloscopes, Communication Testers, Power Meters, Audio Analyzers and so on. The most common mistake users make when they are starting with instrument remote control is, they use **continuous** acquisition mode. If your instrument is in the continuous mode, you are **never** guaranteed proper and repeatable measurement results.

Switch your Spectrum Analyzers to **single** acquisition mode with the SCPI command

```
INIT:CONT OFF
```

For oscilloscopes, use the SCPI command `SING` to start a **single** waveform acquisition.



# 4 Using VXI plug&play Instrument Drivers

Referenced files - all packed into `MATLAB_ICT_rsspecan_Examples.zip`:

- ▶ `MATLAB_ICT_rsspecan_OpenClose.m`
- ▶ `MATLAB_ICT_rsspecan_Open_SetGet_Close.m`
- ▶ `MATLAB_ICT_rsspecan_Complex_Example.m`

Required software. The actual software we used to prepare this chapter is mentioned in the round brackets:

- ▶ MATLAB 2013 or later (2016a)
- ▶ MATLAB Instrument Control Toolbox, further referred to as ICT
- ▶ Windows XP / VISTA / Win 7 (Win 7 64-bit)
- ▶ R&S VISA 5.5.5 or other vendor VISA (R&S VISA 5.5.5)
- ▶ Rohde & Schwarz VXIplug&play instrument driver ([rsspecan VXIplug&play driver 64 bit](#) 3.8.0)
- ▶ for 64-bit MATLAB: Supported compiler. See the list of [Supported compilers](#) (Microsoft Visual C++ 2015 Professional)

## 4.1 Installing VXIplug&play Instrument Drivers

Rohde & Schwarz VXI plug&play instrument drivers are available in the Drivers download area on our website:

[Rohde & Schwarz driver search](#)

After installing the instrument driver, the ICT gives you the option to verify the installation. After the successful installation of `rsspecan` instrument driver, use the following command:

```
>> instrhwinfo ('vxipnp', 'rsspecan')  
  
ans =  
  
HardwareInfo with properties:  
  
Manufacturer: 'Rohde & Schwarz GmbH'  
Model: 'Rohde&Schwarz Spectrum Analyzer'  
DriverVersion: '1.0'  
DriverDllName: 'C:\Program Files\IVI Foundation\VISA\Win64\bin\rsspecan_64.dll'
```

Access to your hardware may be provided by a support package.  
Go to the Support Package Installer to learn more.



The type of VXI plug&play instrument driver always has to match the type of MATLAB, not the type of your operating system. That means:

For MATLAB 64-bit, you can only use 64-bit VXI plug&play instrument drivers.

For MATLAB 32-bit, you can only use 32-bit VXI plug&play instrument drivers even on 64-bit operating system.

## 4.2 MATLAB MDD Drivers

As a part of Rohde & Schwarz VXI plug&play instrument drivers provides MATLAB MDD drivers (single file with mdd extension). For example, you can find the `rsspecan.mdd` file in the driver's directory:

32-bit VXI plug&play instrument driver base path:

```
c:\Program Files (x86)\IVI Foundation\VISA\WinNT\rsspecan
```

64-bit VXI plug&play instrument driver base path:

```
c:\Program Files\IVI Foundation\VISA\Win64\rsspecan
```

MATLAB MDD drivers are XML files that provide an interface between VXI plug&play instrument driver and MATLAB. Rohde & Schwarz VXI plug&play instrument drivers are attribute-based, i.e. you can access instrument's capabilities either with **Functions** or **Attributes** (in MATLAB called **Properties**). They are described in the help file `rsspecan_vxi.chm` located in the same folder as the `rsspecan.mdd` file:

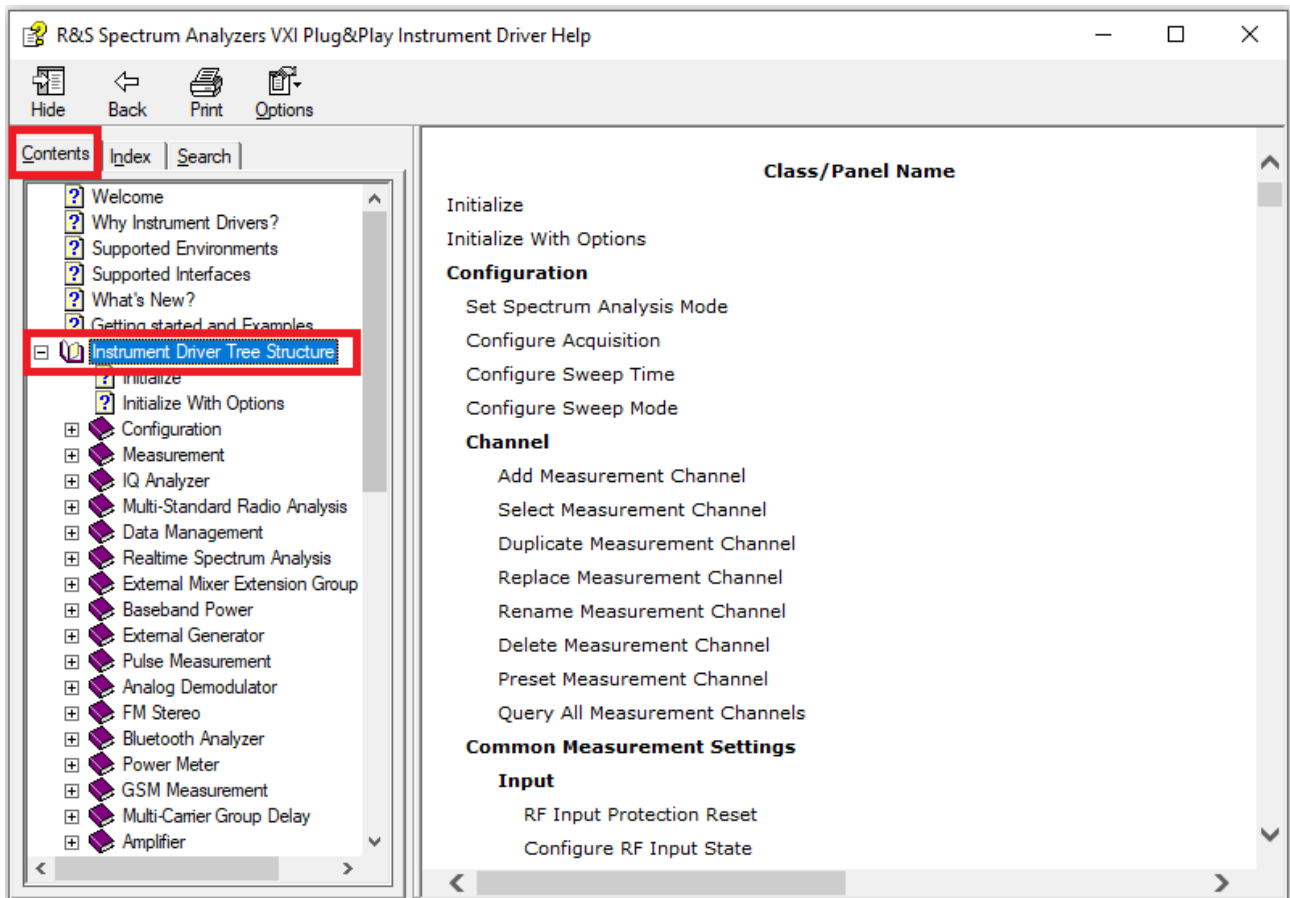


Figure 1: Functions of the VXI plug&play instrument driver described in the \*.chm file

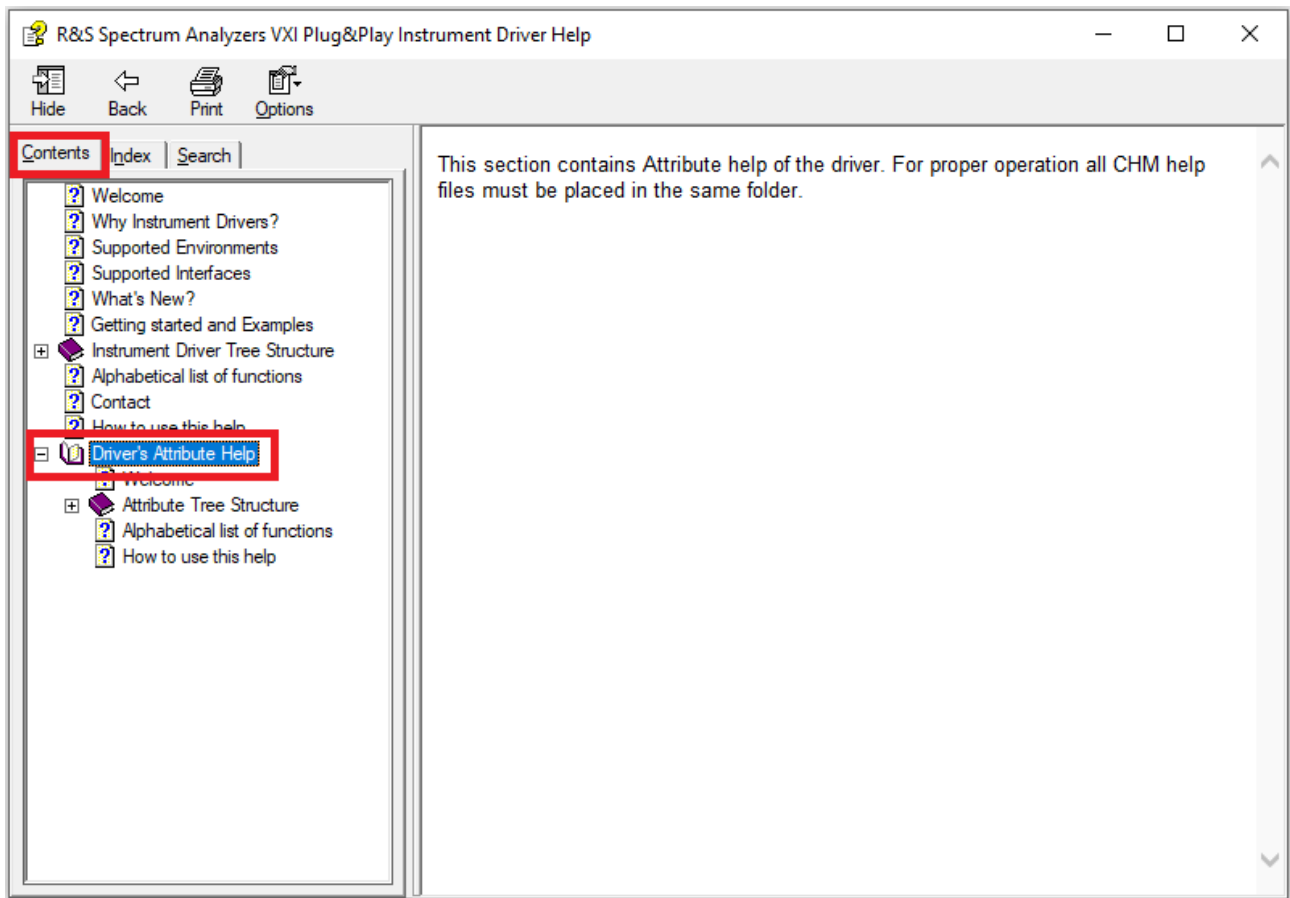


Figure 2: Attributes of the VXI plug&play instrument driver described in the \*.chm file

### 4.3 Opening and Closing Instrument Driver Session

Below is an example of MATLAB script that opens new `rsspecan` session, queries the instrument identification string and closes the session. This script is also available as `MATLAB_ICT_rsspecan_OpenClose.m` :

```
% Create a device object and connect to the instrument
specan = icdevice('rsspecan.mdd', 'TCPIP::192.168.1.100::INSTR');
connect(specan)

% Query ID response
idQueryResponse = zeros (1024, 1);
[idQueryResponse] = invoke (specan, 'IDQueryResponse', 1024, idQueryResponse)

% Disconnect device object from the instrument and delete the object.
disconnect(specan);
delete(specan);
```

## 4.4 Calling Instrument Driver Functions

All the code examples used in next chapters we summarized into the following file:

MATLAB\_ICT\_rsspecan\_Open\_SetGet\_Close.m

Use the help file `rsspecan_vxi.chm` to find a function you want to call. Let us, for example, take setting of the Spectrum Analyzer center frequency. The function help also contains the MATLAB code snippet:  

**R&S Spectrum Analyzers VXI Plug&Play Instrument Driver Help**

Contents | Index | Search

- Initialize With Options
- Configuration
  - Set Spectrum Analysis Mode
  - Configure Acquisition
  - Configure Sweep Time
  - Configure Sweep Mode
- Channel
- Common Measurement Settings
  - Input
  - Output
  - User Port
  - Frequency and Span
    - Configure Frequency Start Stop
    - Configure Frequency Center Op
    - Configure Frequency Center**
    - Configure Frequency Stop Stop
    - Configure Frequency Step Size Auto
    - Configure Horizontal Scale
    - Configure Frequency Span Full
    - Configure Frequency Span Zero
    - Configure Frequency Offset
    - Configure Frequency Coupling Factor
    - Configure Signal Track
    - Configure Frequency Mode
    - Configure Frequency Level
    - Configure Subwindow Horizontal Scale
- Amplitude

### Configure Frequency Center

#### C Function Prototype

```
ViStatus rsspecan_ConfigureFrequencyCenter (ViSession instrumentHandle, ViInt32 reserved, ViReal64 centerFrequency);
```

#### Matlab Prototype

```
invoke (deviceObj, 'ConfigureFrequencyCenter', reserved, centerFrequency)
```

#### Purpose

This function configures the center frequency the spectrum analyzer

Attribute(s):  
RSSPECAN\_ATTR\_FREQUENCY\_CENTER

Remote-control command(s):  
[SENSE:]FREQUENCY:CENTER

#### Parameters

Name	Type	Description
<b>instrumentHandle</b>	ViSession	The ViSession handle that you obtain from the <code>rsspecan_init</code> or <code>rsspecan_InitWithOptions</code> function. The handle identifies a particular instrument session. Default Value: None
<b>reserved</b>	ViInt32	This control is reserved.
<b>centerFrequency</b>	ViReal64	The center frequency of the frequency sweep. Units: Hertz Default Value: 0.0 Attribute: RSSPECAN_ATTR_FREQUENCY_CENTER

Figure 3: VXI plug&play instrument driver functions help including MATLAB prototypes


Copy and paste the part marked   into your code and adjust the parameters.

### Example:

```
invoke (specan, 'ConfigureFrequencyCenter', 0, 110E6)
```

## 4.5 Setting Property Value

Use the help file `rsspecan_vxi.chm` to find a property you want to set. As an example, we use the same instrument parameter as in the previous chapter: center frequency. The property help also contains the MATLAB settings code snippet: 

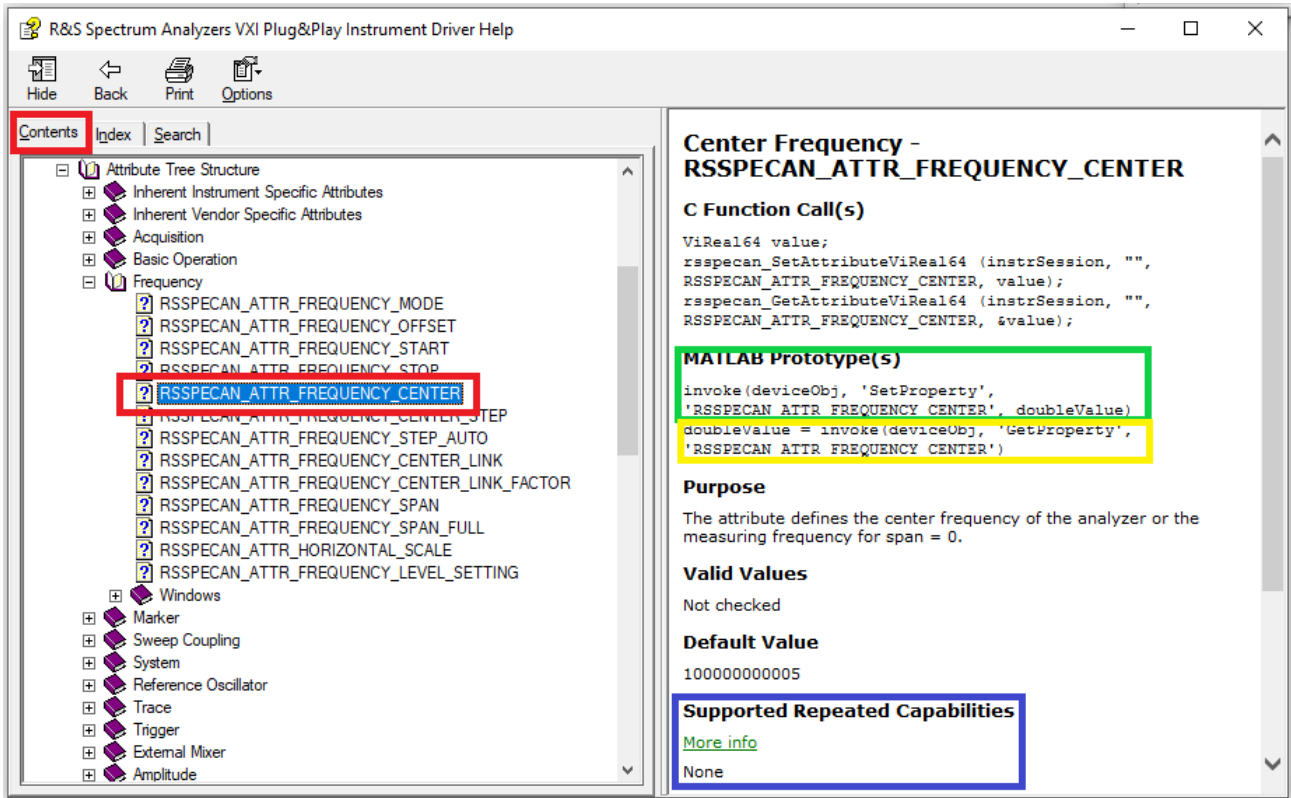




Figure 4: VXI plug&play instrument driver properties help including MATLAB prototypes for setting

Copy and paste the part marked  into your code, adjust the object name and the last two parameters:

`value_to_be_set`: enter the desired frequency as double number.

`repeated_capability_string`: non-mandatory string parameter. See the part marked  **Supported Repeated Capabilities** for valid values. In our case, the attribute `RSSPECAN_ATTR_FREQUENCY_CENTER` has no repeated capabilities supported, therefore you can leave this parameter out. The chapter **Repeated Capabilities** describes this parameter in more details.

**Example:**



```
invoke(specan, 'SetProperty', 'RSSPECAN_ATTR_FREQUENCY_CENTER', 110E6)
```

## 4.6 Reading Property Value

Copy and paste the part marked  into your code and adjust the object name.

`repeated_capability_string`: non-mandatory string parameter, same as for setting the property value.

**Example:**



```
centerFrequency = invoke(specan, 'GetProperty', 'RSSPECAN_ATTR_FREQUENCY_CENTER')
```

## 4.7 Repeated Capabilities

To briefly explain Repeated Capabilities, let us take the property `RSSPECAN_ATTR_MARKER_ENABLED` as an example: This property has boolean value of True or False (Marker State ON or OFF). However, you also need to communicate to the instrument which one of the sixteen markers (the area marked  ) you want to enable. This information you enter as `repeated_capability_string`. If a property have more Repeated Capabilities defined, they are separated by commas without spaces. An example of such property is `RSSPECAN_ATTR_TRACE_STATE`.

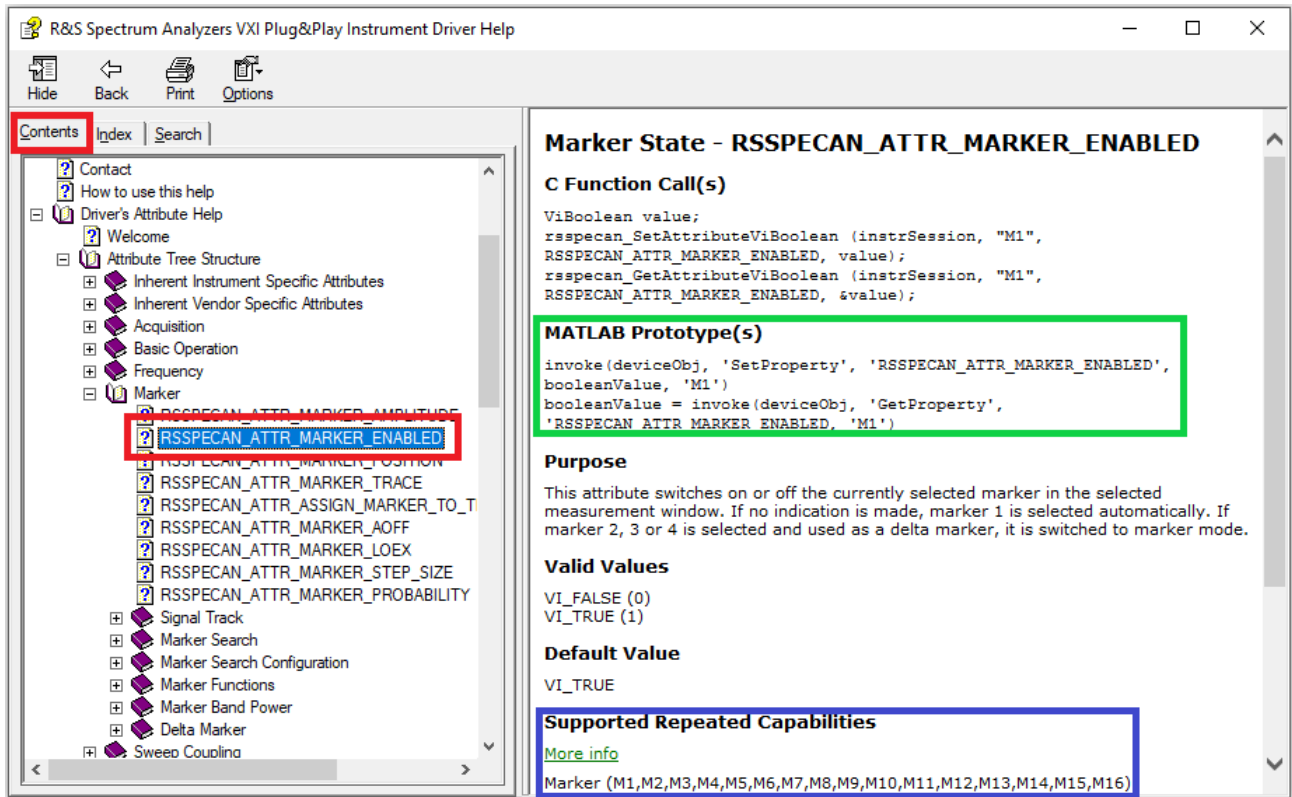


Figure 5: Marker property with defined Repeated Capabilities

**Example:**

```
 
```

```
invoke(specan, 'SetProperty', 'RSSPECAN_ATTR_MARKER_ENABLED', 1, 'M1')  
invoke(specan, 'SetProperty', 'RSSPECAN_ATTR_TRACE_STATE', 1, 'Win1,TR1')
```

## 4.8 Property Identifier

In the previous chapters when setting and reading properties, we used the following property identifier string for center frequency: `RSSPECAN_ATTR_FREQUENCY_CENTER`. However, there are two names and their variants you can use as well - see the texts marked   in the help file screenshot below:

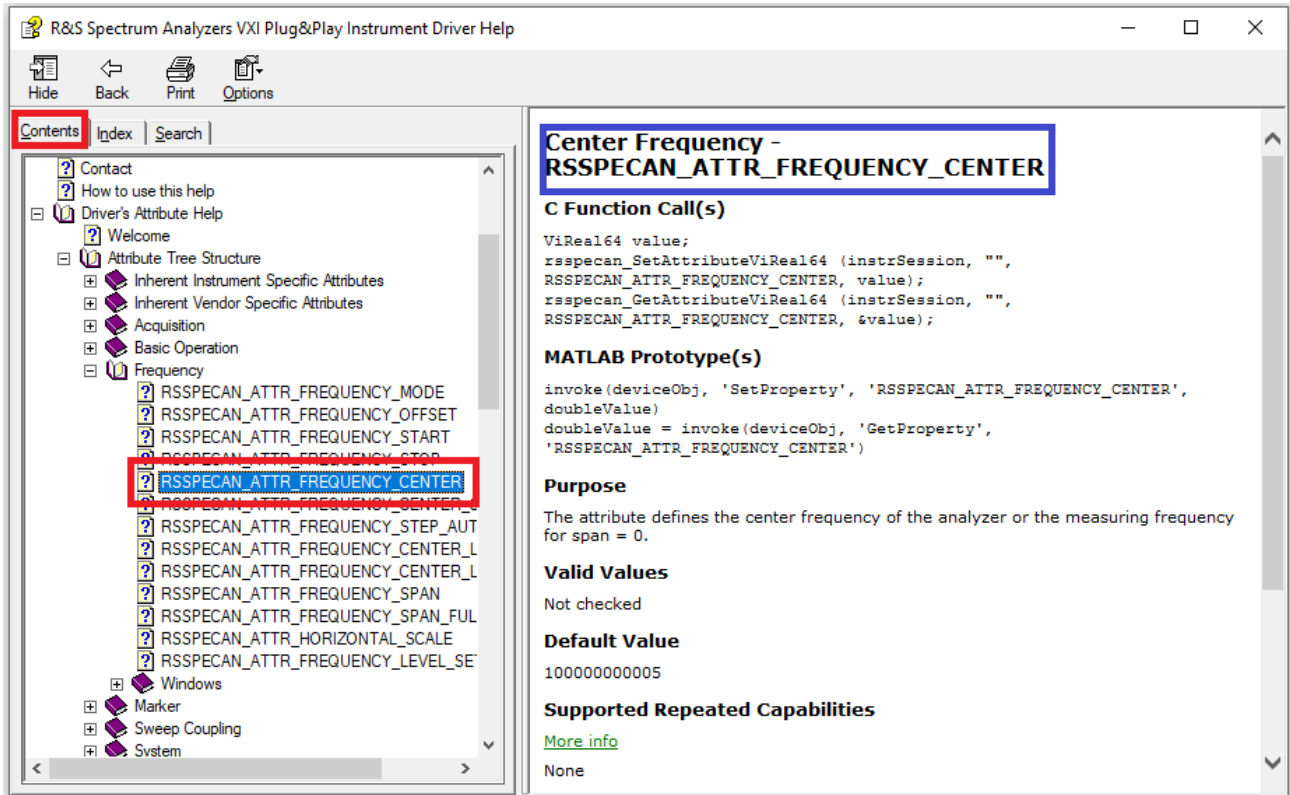


Figure 6: Property Identifiers in the rsspecan help file

All the invoke calls below achieve the same action, they only differ in Property ID:

```
% Complete Property ID
invoke(specan, 'SetProperty', 'RSSPECAN_ATTR_FREQUENCY_CENTER', 110E6)

% Property ID without prefix
invoke(specan, 'SetProperty', 'ATTR_FREQUENCY_CENTER', 110E6)

% Property ID case Insensitive
invoke(specan, 'SetProperty', 'RsSPeCAN_Attr_FrEQuENcY_CeNtEr', 110E6)

% Descriptive name
invoke(specan, 'SetProperty', 'Center Frequency', 110E6)

% Descriptive name with underscores
invoke(specan, 'SetProperty', 'Center_Frequency', 110E6)

% Descriptive name case insensitive
invoke(specan, 'SetProperty', 'CeNtEr FrEqUeNcY', 110E6)
```