# How to use Rohde & Schwarz Instruments in MATLAB
Application Note

**Products:**

| Rohde & Schwarz VXIplug&play Instrument Drivers

This application note outlines different approaches for remote-controlling Rohde & Schwarz instruments out of MathWorks MATLAB. For this purpose the Rohde & Schwarz VXIplug&play instrument drivers are used.

ROHDE&SCHWARZ

Application Note

Miloslav Macko
20-May-15-1MA171_10e

# Table of Contents

# 1 Preface

This application note presents methods for integrating Rohde & Schwarz test and measurement (T&M) instruments into The MathWorks MATLAB applications. This allows you to remote-control Rohde & Schwarz instruments for T&M applications from MATLAB .

Please note, that the focus of this application note is an approach to communicate with Rohde & Schwarz instruments using instrument drivers. However, a chapter 2.5 is dedicated to plain SCPI communication over VISA also for the purpose of testing the connection to the instrument from MATLAB .

***MATLAB_VISA_Interface_help***
***MATLAB_VISA_ReadWrite_help***

For demonstration purposes the Rohde & Schwarz VXIplug&play instrument drivers for spectrum analyzers (***rsspecan***) is used in this application note. The presented procedure is applicable to all Rohde & Schwarz VXIplug&play instrument drivers. Spectrum analyzer driver is chosen because it represents the best tasks that need to be performed when communicating with instrument – settings, waiting for the measurement result, reading the results either in strings or arrays of numbers.

To illustrate the usage of instrument drivers in MATLAB two approaches are outlined. The first part of this application note describes the MATLAB Instrument Control Toolbox (in this application note further referred to as Test & Measurement tool or shortly TMTOOL) as a high-level approach with tool support, and error handling. Besides this, a low-level approach using the MATLAB external interface *calllib* is explained in the second part of this application note.

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

National Instruments are U.S. registered trademarks of National Instruments.

MATLAB is a registered trademark of The MathWorks, Inc.

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

## 1.1 Related documents

The following application note discusses remote-control instrument drivers and their usage:

- ***1GP69: R&S NRP-Z Power Sensor Programming Guide***
  The R&S NRP-Z power sensors from Rohde & Schwarz represent the latest in power measurement technology. They offer all the functionality of conventional power meters, and more, within the small housing of a power sensor. This application note serves as a coding guide for situations in which the R&SNRP-Z power sensors are to be used in custom test and measurement software.

- ***1MA153: Development Hints and Best Practices for Using Instrument Drivers***
  The aim of this paper is to provide information regarding Rohde & Schwarz instrument drivers. This paper shall help application engineers, as well as software developers to easily get an understanding of advanced techniques to develop test and measurement (T&M) applications by utilizing Rohde & Schwarz instrument drivers. Furthermore the nomenclature used for Rohde & Schwarz instrument drivers will be explained.

- ***1EF62: Hints and Tricks for Remote Control of Spectrum and Network Analyzers***
  This application note provides hints for implementing remote control programs using Rohde & Schwarz spectrum and network analyzers. The document makes suggestions for improved remote control performance and describes aspects of measurement synchronization in detail. Finally the document discusses some typical challenges of remote control in production test.

- ***1GP60: R&S MATLAB Toolkit for Signal Generators and Power Sensors***
  The R&S MATLAB Toolkit for signal generators and power sensors provides routines for remote-controlling these instruments. Additional MATLAB scripts turn I/Q vectors into the Rohde & Schwarz waveform generator file format for use with an ARB. This application note describes the installation and use of the R&S MATLAB Toolkit on Microsoft Windows and Linux based systems.

## 1.2 Required Software

To follow the configuration steps described in this application note the following software is needed for the 32-bit application examples:

- MATLAB 2013a or later
- Windows XP/VISTA/7 32-bit operating system, >2Gbyte RAM
- VISA I/O library (e.g. National Instruments VISA Version 5.x)
- Rohde & Schwarz VXIplug&play instrument driver 32-bit support
- MATLAB Instrument control toolbox for chapters 2 and 3

To follow the 64-bit examples the following software is needed

- MATLAB 2013a or later
- Windows VISTA/7 64-bit operating system
- VISA I/O library (e.g. National Instruments VISA Version 5.x)
- Rohde & Schwarz VXIplug&play instrument driver with 64-bit support
- MATLAB supported compiler. See the list of supported compilers here: ***http://www.mathworks.de/support/compilers/current_release/win64.html***.For example: Microsoft Visual C++ 2008 SP1 V9.0 Professional Edition compiler
- MATLAB Instrument control toolbox for chapters 2 and 3

## 1.3 About MATLAB Instrument Toolbox

Instrument Control Toolbox™ lets you connect MATLAB directly to instruments such as oscilloscopes, function generators, signal analyzers, power supplies, and analytical instruments. The toolbox connects to your instruments via instrument drivers such as IVI and VXIplug&play, or via text-based SCPI commands over commonly used communication protocols such as GPIB, VISA, TCP/IP, and UDP. You can also control and acquire data from your test equipment without writing code.

# 2 Using the TMTOOL

The MATLAB TMTOOL extends MATLAB to remote-control test or measurement equipment via GPIB (IEEE 488.2), TCP/IP (VXI-11, …), etc. This makes it possible to transfer data, e.g. writing instrument settings or transferring generated I/Q waveform files from the instrument to the PC via various interfaces.

The TMTOOL offers tools for generating MATLAB code out of the VXIplug&play instrument driver. The TMTOOL supports IVI and VXIplug&play instrument drivers. This application note focuses on the latter case. To utilize VXIplug&play instrument drivers, the TMTOOL generates a MATLAB instrument driver file out of the VXIplug&play instrument driver's DLL. (wrapper around dll functions). This step is described in chapter 2.2; wrapper code can be seen in chapter 2.3.

**Prerequisites**

The MATLAB command *instrhwinfo* enables you to verify a proper recognition of your installed VISA library. Having `'visa'` among supported interfaces is mandatory for this application note (***NI_VISA_Download_link***). Your command window output should look like this:

```
>> instrhwinfo

ans =

          MATLABVersion: '8.1 (R2013a)'
    SupportedInterfaces: {'gpib'    'serial'    'tcpip'    'udp'
'visa'   'Bluetooth'   'i2c'}
       SupportedDrivers: {'matlab'   'ivi'   'vxipnp'}
            ToolboxName: 'Instrument Control Toolbox'
         ToolboxVersion: '3.3 (R2013a)'
```

Further information can be gathered by querying your VISA details:

```
>> instrhwinfo ('visa')

ans =
 InstalledAdaptors: {'ni'}
 JarFileVersion: 'Version 3.3'
```

## 2.1 Installing VXIplug&play Instrument Drivers

Installing the VXIplug&play instrument driver on the host PC is a prerequisite before creating the MATLAB instrument driver file (extension: .mdd). This and the following chapter will guide through that process.

Rohde & Schwarz VXIplug&play instrument drivers and installation manuals are available in the Drivers download area on the Rohde & Schwarz website: ***http://www.rohde-schwarz.com/en/search/driver_63451.html?term=****.

The version of VXIplug&play driver always has to match the version of MATLAB , not the version of operating system. That means, for 32-bit MATLAB only 32-bit version of VXIplug&play driver can be used and for 64-bit MATLAB only 64-bit version of VXIplug&play driver can be used. To avoid the mix-up it is recommended to install only one version of VXIplug&play driver.

More specifically, the VXIplug&play driver discussed in this application note can be found here ***http://www.rohde-schwarz.com/en/driver/fsw***

After installing your specific VXIplug&play instrument driver, the TMTOOL gives you a possibility to verify the installation (see chapter 2.3). The installation can also be verified via the command *instrhwinfo ('vxipnp', 'INSTRUMENT_DRIVER')*, where *INSTRUMENT_DRIVER* is the driver name. For example, a successfully installed *rsspecan* instrument driver can look like this:

```
>> instrhwinfo ('vxipnp', 'rsspecan')
ans =
 Manufacturer: 'Rohde & Schwarz GmbH'
 Model: 'Rohde&Schwarz Spectrum Analyzer'
 DriverVersion: '1.0'
 DriverDllName: 'C:\Program Files (x86)\IVI
Foundation\VISA\WINNT\bin\rsspecan_32.dll'
```

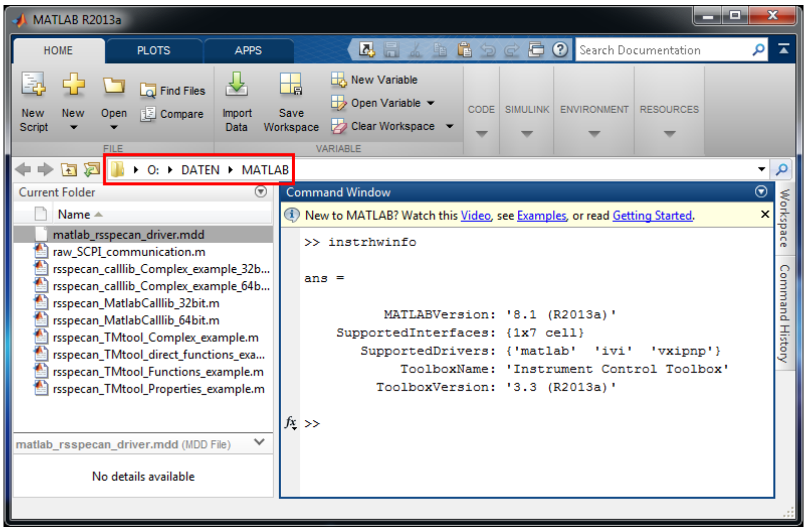
## 2.2 Creating MATLAB Instrument Drivers

The command *makemid ('driver', 'filename')* creates a MATLAB instrument driver from the instrument driver's DLL description[1], where 'driver' is the original VXIplug&play driver name identified by *instrhwinfo* or the TMTOOL (see chapter 2.1). For example, the *rsspecan* instrument driver can be created using the following command:

```
>> makemid ('rsspecan', 'matlab_rsspecan_driver')
```

**Note:** Chapters 2.6 and 2.7 describe the usage of MATLAB universal driver created by `makemid`. However, Rohde & Schwarz provides its custom made MATLAB instrument driver file (further referred as mdd driver) which perfectly fits the VXIplug&play driver structure and is completely in line with instrument driver help files. In all new instrument driver releases help files will even contain snippets of MATLAB code for simple copy and paste to MATLAB scripts. Refer to chapter 3: Working with R&S custom made drivers to see the advantages of using R&S custom built mdd driver: simple structure (which means faster development), flexibility, better code readability and all informations found in one place: R&S Instrument drivers help file. In addition, ready-to-use MATLAB script examples for rsspecan (CUSTOM_rsspecan_TMTOOL_Complex_example.m) and rsnrpz (CUSTOM_rsnrpz_TMTOOL_example.m) are provided to see the usage of R&S custom mdd drivers in praxis.

In the current working directory, the MATLAB instrument driver file (extension: .mdd) will be created from the specified VXIplug&play instrument driver. Current working directory is shown in upper part of MATLAB GUI:



matlab_rsspecan_driver.mdd
raw_SCPI_communication.m
rsspecan_calllib_Complex_example_32b...
rsspecan_calllib_Complex_example_64b...
rsspecan_MatlabCalllib_32bit.m
rsspecan_MatlabCalllib_64bit.m
rsspecan_TMtool_Complex_example.m
rsspecan_TMtool_direct_functions_exa...
rsspecan_TMtool_Functions_example.m
rsspecan_TMtool_Properties_example.m

matlab_rsspecan_driver.mdd (MDD File)

No details available

New to MATLAB? Watch this Video, see Examples, or read Getting Started.

```
>> instrhwinfo

ans =

         MATLABVersion: '8.1 (R2013a)'
   SupportedInterfaces: {1x7 cell}
      SupportedDrivers: {'matlab'  'ivi'  'vxipnp'}
           ToolboxName: 'Instrument Control Toolbox'
        ToolboxVersion: '3.3 (R2013a)'

>>
```

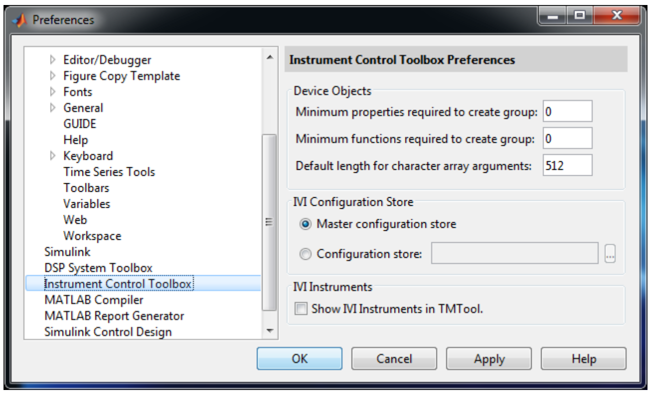
***Figure 1: MATLAB default path field***

[1] The TMTOOL uses the API information of the VXIplug&play instrument driver stored in the VXIplug&play function panel (extension: .fp) file.

If you wish to specify a different target path, use the *makemid ('driver', 'filepath')*. For example in MATLAB 32-bit:

```
>> makemid ('rsspecan', 'c:\Program Files
(x86)\MATLAB\R2013a\32\toolbox\instrument\instrument\drivers\mat
lab_rsspecan_driver')
```

Settings for generation of the driver can be change in ***Prefrerences -> Instrument Control Toolbox***:



Instrument Control Toolbox Preferences

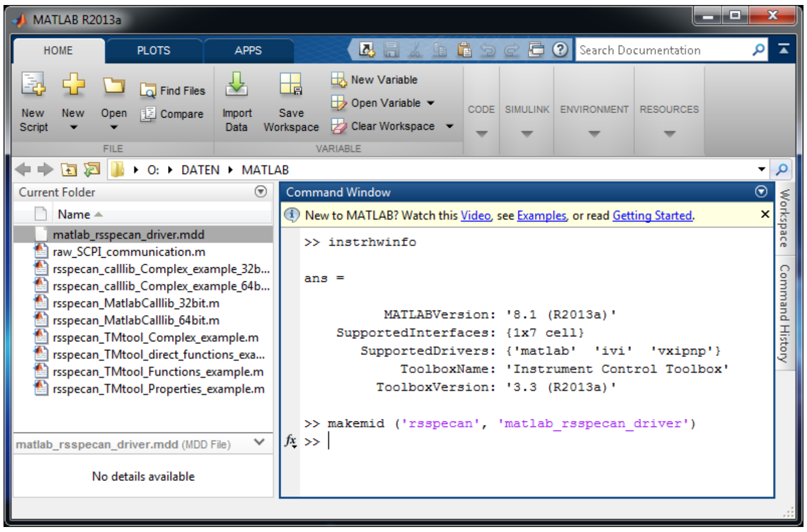
Device Objects
Minimum properties required to create group: 0
Minimum functions required to create group: 0
Default length for character array arguments: 512

IVI Configuration Store
(●) Master configuration store
( ) Configuration store:

IVI Instruments
[ ] Show IVI Instruments in TMTool.

*Figure 2: Preferences for MATLAB Instrument Control Toolbox*

Command creates the rsspecan driver in the defined path with defined name.
Note: You always have to define the name of target MATLAB instrument driver file, otherwise no driver will be created. Extension is discarded and always replaced with *.mdd. The path mentioned in the last example is the default path where all MATLAB instrument driver files are usually located.


MATLAB R2013a
HOME
PLOTS
APPS
Search Documentation
New Script
New
Open
Find Files
Compare
FILE
Import Data
Save Workspace
New Variable
Open Variable
Clear Workspace
VARIABLE
CODE
SIMULINK
ENVIRONMENT
RESOURCES
O: ▸ DATEN ▸ MATLAB
Current Folder
Name
matlab_rsspecan_driver.mdd
raw_SCPI_communication.m
rsspecan_calllib_Complex_example_32b...
rsspecan_calllib_Complex_example_64b...
rsspecan_MatlabCalllib_32bit.m
rsspecan_MatlabCalllib_64bit.m
rsspecan_TMtool_Complex_example.m
rsspecan_TMtool_direct_functions_exa...
rsspecan_TMtool_Functions_example.m
rsspecan_TMtool_Properties_example.m
matlab_rsspecan_driver.mdd (MDD File)
No details available
Command Window
New to MATLAB? Watch this Video, see Examples, or read Getting Started.
Workspace
Command History

```
>> instrhwinfo

ans =

        MATLABVersion: '8.1 (R2013a)'
  SupportedInterfaces: {1x7 cell}
     SupportedDrivers: {'matlab'  'ivi'  'vxipnp'}
          ToolboxName: 'Instrument Control Toolbox'
       ToolboxVersion: '3.3 (R2013a)'

>> makemid ('rsspecan', 'matlab_rsspecan_driver')
>> 
```

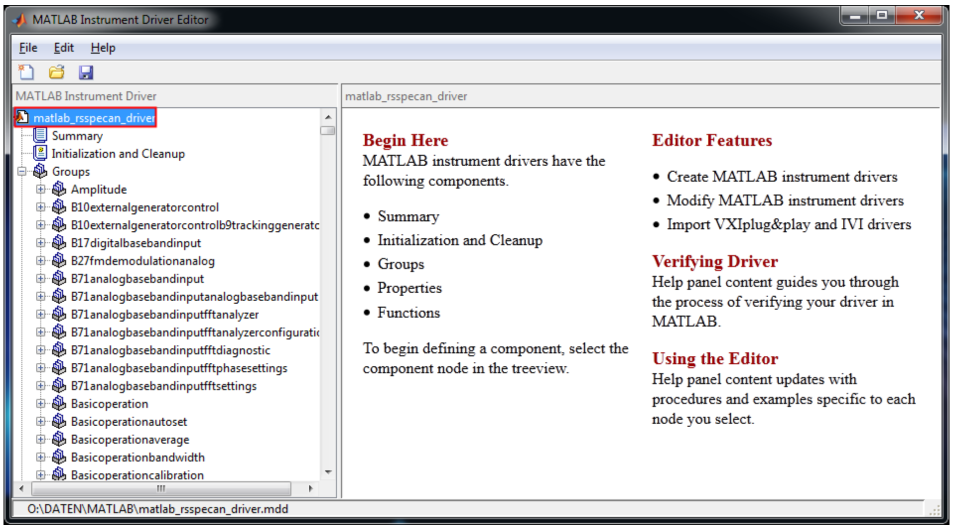
*Figure 3: Creating a MATLAB instrument driver from a VXIplug&play instrument driver.*

Please do not uninstall your instrument driver after this step. The instrument driver is accessed by MATLAB when communicating with your connected instrument.

## 2.3 Instrument Driver Editor

Double-clicking on mdd file or using command *midedit ('driver_name')* opens Instrument Driver Editor for that MATLAB driver. Using *midedit* without additional parameters opens new untitled driver for editing.

```
>> midedit('matlab_rsspecan_driver')
```



MATLAB Instrument Driver Editor

File Edit Help

MATLAB Instrument Driver

- matlab_rsspecan_driver
  - Summary
  - Initialization and Cleanup
  - Groups
    - Amplitude
    - B10externalgeneratorcontrol
    - B10externalgeneratorcontrolb9trackinggeneratc
    - B17digitalbasebandinput
    - B27fmdemodulationanalog
    - B71analogbasebandinput
    - B71analogbasebandinputanalogbasebandinput
    - B71analogbasebandinputfftanalyzer
    - B71analogbasebandinputfftanalyzerconfiguratic
    - B71analogbasebandinputfftdiagnostic
    - B71analogbasebandinputfftphasesettings
    - B71analogbasebandinputfftsettings
    - Basicoperation
    - Basicoperationautoset
    - Basicoperationaverage
    - Basicoperationbandwidth
    - Basicoperationcalibration

matlab_rsspecan_driver

**Begin Here**
MATLAB instrument drivers have the following components.

- Summary
- Initialization and Cleanup
- Groups
- Properties
- Functions

To begin defining a component, select the component node in the treeview.

**Editor Features**

- Create MATLAB instrument drivers
- Modify MATLAB instrument drivers
- Import VXIplug&play and IVI drivers

**Verifying Driver**
Help panel content guides you through the process of verifying your driver in MATLAB.

**Using the Editor**
Help panel content updates with procedures and examples specific to each node you select.

O:\DATEN\MATLAB\matlab_rsspecan_driver.mdd

***Figure 4: MATLAB Instrument Driver Editor Window.***

This editor allows you to customize and modify your generated MATLAB instrument driver. But it can also serve as a very good development tool, since searching options of TMTOOL are limited. Refer to the chapter 2.7 to see how Instrument Driver Editor can be utilized to create the MATLAB script effectively. For further information about this editor, please consult the MATLAB TMTOOL documentation.

## 2.4 Test & Measurement Tool

The MATLAB TMTOOL allows you to explore created MATLAB instrument drivers and connected test or measurement equipment (see Figure 6). For example browsing the *rsspecan* VXIplug&play instrument driver is shown on Figure 7.

Instrument Control Toolbox can be started by command *tmtool* or using APPS tab icon:

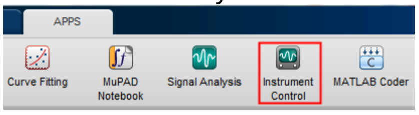
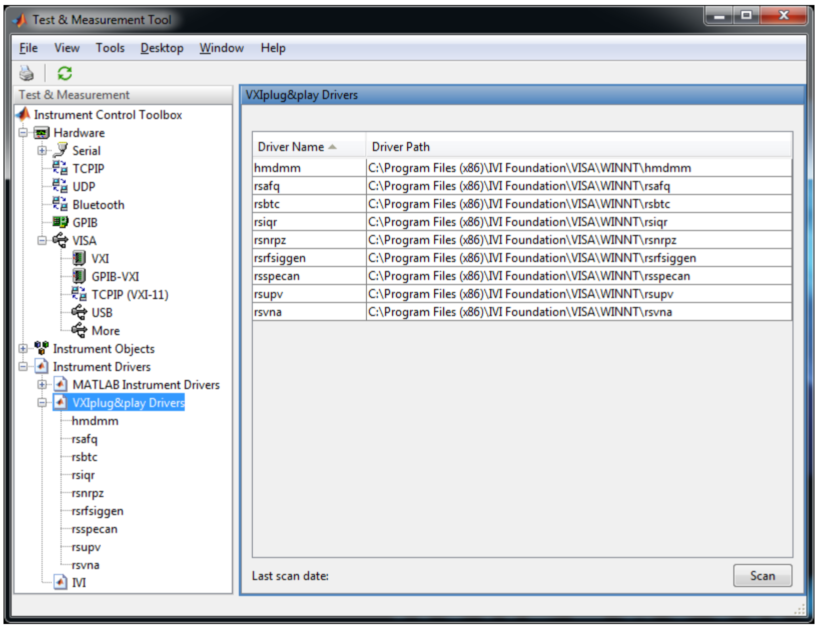


***Figure 5: TMTOOL placement on MATLAB APPS bar***

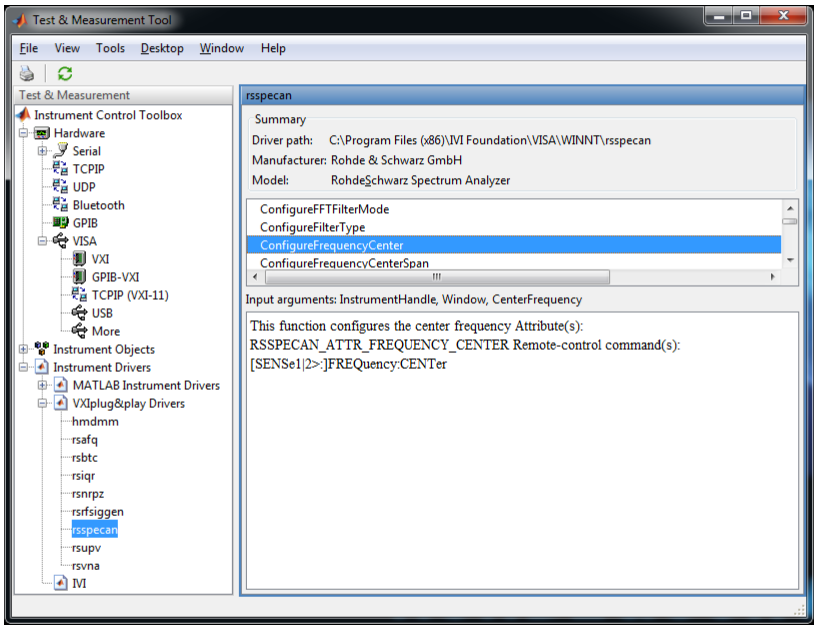
```
>> tmtool
```



| Driver Name | Driver Path |
|---|---|
| hmdmm | C:\Program Files (x86)\IVI Foundation\VISA\WINNT\hmdmm |
| rsafq | C:\Program Files (x86)\IVI Foundation\VISA\WINNT\rsafq |
| rsbtc | C:\Program Files (x86)\IVI Foundation\VISA\WINNT\rsbtc |
| rsiqr | C:\Program Files (x86)\IVI Foundation\VISA\WINNT\rsiqr |
| rsnrpz | C:\Program Files (x86)\IVI Foundation\VISA\WINNT\rsnrpz |
| rsrfsiggen | C:\Program Files (x86)\IVI Foundation\VISA\WINNT\rsrfsiggen |
| rsspecan | C:\Program Files (x86)\IVI Foundation\VISA\WINNT\rsspecan |
| rsupv | C:\Program Files (x86)\IVI Foundation\VISA\WINNT\rsupv |
| rsvna | C:\Program Files (x86)\IVI Foundation\VISA\WINNT\rsvna |

***Figure 6: Installed VXIplug&play instrument drivers shown in the MATLAB TMTOOL.***

A MATLAB driver can be created from these VXIplug&play drivers as described in chapter 2.2. Also the VXIplug&play driver help information is accessible using the TMTOOL, as shown in the figure below (Figure 7) on example of the ***rsspecan*** function ***ConfigureFrequencyCenter***:



Driver path: C:\Program Files (x86)\IVI Foundation\VISA\WINNT\rsspecan
Manufacturer: Rohde & Schwarz GmbH
Model: RohdeSchwarz Spectrum Analyzer

ConfigureFFTFilterMode
ConfigureFilterType
ConfigureFrequencyCenter
ConfigureFrequencyCenterSpan

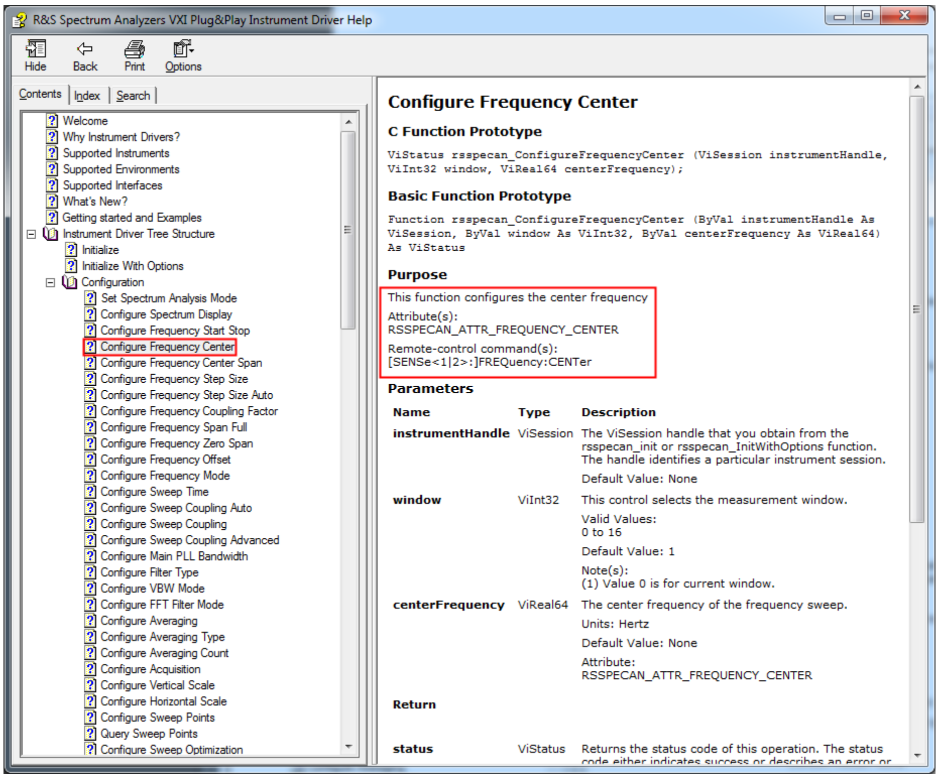
Input arguments: InstrumentHandle, Window, CenterFrequency

This function configures the center frequency Attribute(s):
RSSPECAN_ATTR_FREQUENCY_CENTER Remote-control command(s):
[SENSe1|2>:]FREQuency:CENTer

***Figure 7: Description of API function ConfigureFrequencyCenter of the VXIplug&play instrument driver.***

### 2.4.1 Instrument Drivers Extended Help

All Rohde & Schwarz instrument drivers come with detailed help file which is located in the same directory as *.fp file. In our example of rsspecan driver it would be "**c:\Program Files (x86)\IVI Foundation\VISA\WinNT\rsspecan\rsspecan_vxi.chm**". (for 64-bit VXI plug&play driver the path is "**c:\Program Files\IVI Foundation\VISA\Win64\rsspecan\rsspecan_vxi.chm**")
The detailed help for function ***rsspecan_ConfigureFrequencyCenter*** is shown on Figure 8:


R&S Spectrum Analyzers VXI Plug&Play Instrument Driver Help
Hide
Back
Print
Options
Contents
Index
Search
Welcome
Why Instrument Drivers?
Supported Instruments
Supported Environments
Supported Interfaces
What's New?
Getting started and Examples
Instrument Driver Tree Structure
Initialize
Initialize With Options
Configuration
Set Spectrum Analysis Mode
Configure Spectrum Display
Configure Frequency Start Stop
Configure Frequency Center
Configure Frequency Center Span
Configure Frequency Step Size
Configure Frequency Step Size Auto
Configure Frequency Coupling Factor
Configure Frequency Span Full
Configure Frequency Zero Span
Configure Frequency Offset
Configure Frequency Mode
Configure Sweep Time
Configure Sweep Coupling Auto
Configure Sweep Coupling
Configure Sweep Coupling Advanced
Configure Main PLL Bandwidth
Configure Filter Type
Configure VBW Mode
Configure FFT Filter Mode
Configure Averaging
Configure Averaging Type
Configure Averaging Count
Configure Acquisition
Configure Vertical Scale
Configure Horizontal Scale
Configure Sweep Points
Query Sweep Points
Configure Sweep Optimization

**Configure Frequency Center**

**C Function Prototype**

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

**Basic Function Prototype**

```
Function rsspecan_ConfigureFrequencyCenter (ByVal instrumentHandle As
ViSession, ByVal window As ViInt32, ByVal centerFrequency As ViReal64)
As ViStatus
```

**Purpose**

This function configures the center frequency

Attribute(s):
RSSPECAN_ATTR_FREQUENCY_CENTER

Remote-control command(s):
[SENSe<1|2>:]FREQuency:CENTer

**Parameters**

| Name | Type | Description |
|---|---|---|
| **instrumentHandle** | ViSession | The ViSession handle that you obtain from the rsspecan_init or rsspecan_InitWithOptions function. The handle identifies a particular instrument session.<br>Default Value: None |
| **window** | ViInt32 | This control selects the measurement window.<br>Valid Values:<br>0 to 16<br>Default Value: 1<br>Note(s):<br>(1) Value 0 is for current window. |
| **centerFrequency** | ViReal64 | The center frequency of the frequency sweep.<br>Units: Hertz<br>Default Value: None<br>Attribute:<br>RSSPECAN_ATTR_FREQUENCY_CENTER |

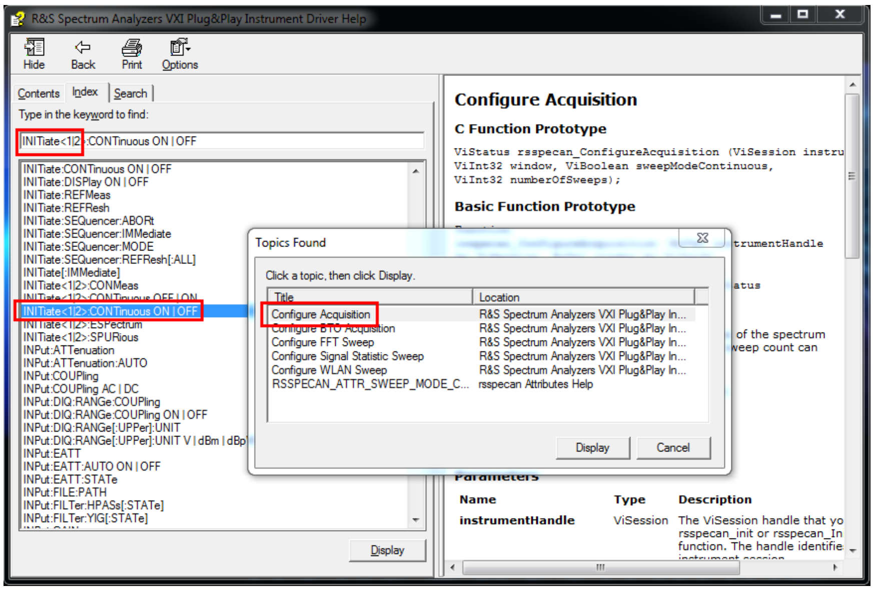
**Return**

| | | |
|---|---|---|
| **status** | ViStatus | Returns the status code of this operation. The status code either indicates success or describes an error or |

***Figure 8: Detailed help description for the function rsspecan_ConfigureFrequencyCenter.***

As you can see, MATLAB TMTOOL shows the section Purpose of this extended help. In the next chapter we will see how these VXIplug&play driver functions have equivalent in MATLAB instrument drivers.

Hint:
If you know the SCPI command and you're looking for a function that utilizes it, use the Index or Search tab and navigate to the SCPI command you're looking for. For example, the SCPI command to set the analyzer to single sweep is "**INITiate:CONTinuous OFF**" - see the Figure 9.



Type in the keyword to find:

INITiate<1|2>:CONTinuous ON | OFF

```
INITiate:CONTinuous ON | OFF
INITiate:DISPlay ON | OFF
INITiate:REFMeas
INITiate:REFResh
INITiate:SEQuencer:ABORt
INITiate:SEQuencer:IMMediate
INITiate:SEQuencer:MODE
INITiate:SEQuencer:REFResh[:ALL]
INITiate[:IMMediate]
INITiate<1|2>:CONMeas
INITiate<1|2>:CONTinuous OFF | ON
INITiate<1|2>:CONTinuous ON | OFF
INITiate<1|2>:ESPectrum
INITiate<1|2>:SPURious
INPut:ATTenuation
INPut:ATTenuation:AUTO
INPut:COUPling
INPut:COUPling AC | DC
INPut:DIQ:RANGe:COUPling
INPut:DIQ:RANGe:COUPling ON | OFF
INPut:DIQ:RANGe[:UPPer]:UNIT
INPut:DIQ:RANGe[:UPPer]:UNIT V | dBm | dBp
INPut:EATT
INPut:EATT:AUTO ON | OFF
INPut:EATT:STATe
INPut:FILE:PATH
INPut:FILTer:HPASs[:STATe]
INPut:FILTer:YIG[:STATe]
```

Topics Found

Click a topic, then click Display.

| Title | Location |
|---|---|
| Configure Acquisition | R&S Spectrum Analyzers VXI Plug&Play In... |
| Configure BTO Acquisition | R&S Spectrum Analyzers VXI Plug&Play In... |
| Configure FFT Sweep | R&S Spectrum Analyzers VXI Plug&Play In... |
| Configure Signal Statistic Sweep | R&S Spectrum Analyzers VXI Plug&Play In... |
| Configure WLAN Sweep | R&S Spectrum Analyzers VXI Plug&Play In... |
| RSSPECAN_ATTR_SWEEP_MODE_C... | rsspecan Attributes Help |

Configure Acquisition

C Function Prototype

```
ViStatus rsspecan_ConfigureAcquisition (ViSession instru
ViInt32 window, ViBoolean sweepModeContinuous,
ViInt32 numberOfSweeps);
```

Basic Function Prototype

trumentHandle

atus

of the spectrum
weep count can

Parameters

| Name | Type | Description |
|---|---|---|
| instrumentHandle | ViSession | The ViSession handle that yo rsspecan_init or rsspecan_In function. The handle identifie |

*Figure 9: Using Rohde & Schwarz  instrument drivers help file cross-references between SCPI commands and VXIplug&play Instrument driver functions.*

Double clicking on indexed item either directly navigates to the appropriate topic, or in case of more than one topics additional pop-up window is displayed. There, you can find the function ***rsspecan_ConfigureAcquisition*** that represents the desired functionality. This name although not very intuitive comes from the IVI specification of setting the sweep mode and number of sweeps.
You can also see that one of the topics is from rsspecan Attributes Help and it points directly to the attribute ***RSSPECAN_ATTR_SWEEP_MODE_CONTINUOUS*** that can be used to set the analyzer to single sweep without having to set the number of sweeps. Refer to 2.6 or 2.7 to see the difference between Functions and Properties (Attributes) approach when using the VXIplug&play instrument driver.

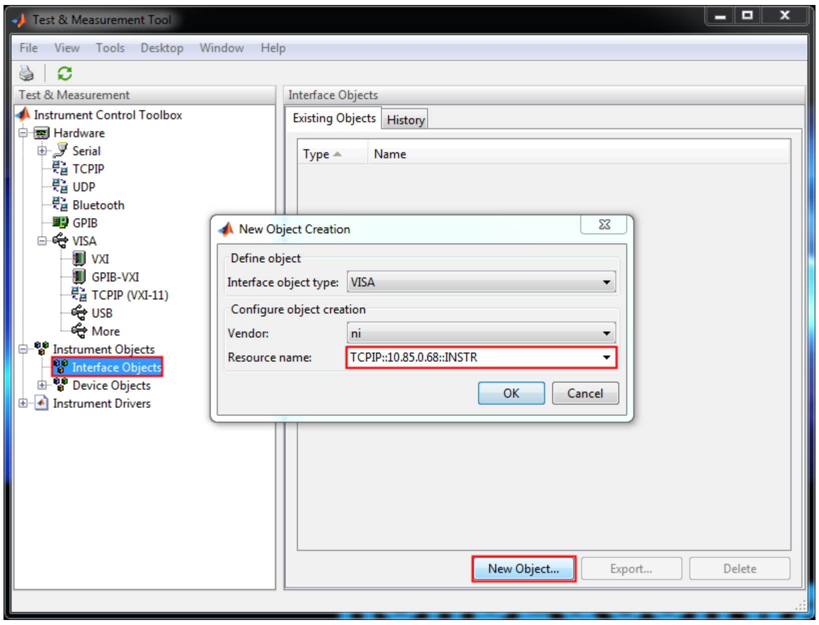
## 2.5 Raw SCPI communication over VISA

This chapter will briefly describe how to remotely communicate with Rohde & Schwarz instruments by not using instrument drivers, but sending directly SCPI commands and receiving responses. It can also serve as a first test that the connection to our instrument from MATLAB can be established. For more help on this topic please refer to following MATLAB help links:

***MATLAB_VISA_Interface_help***
***MATLAB_VISA_ReadWrite_help***

In the TMTOOL left window select ***Instrument Objects -> Interface Objects*** and press ***New Object****…* button (see Figure 10). Then, select ***Interface object type*** VISA, and ***Vendor*** ni (National Instruments). ***VISA Resource name*** is described in the following paragraph:

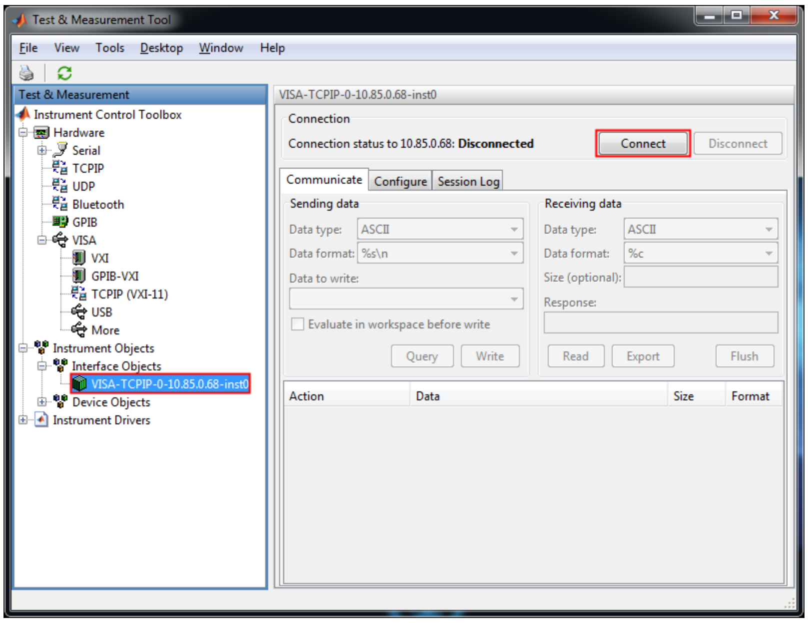
VISA Resource name is a string that specifies the resource to which VISA session will be opened. In our case it is device connected over LAN interface on IP address 10.85.0.68. Therefore VISA Resource name is TCPIP::10.85.0.68::INSTR.
For more information about VISA resource names refer to ***1MA153: Development Hints and Best Practices for Using Instrument Drivers*** or NI online help ***VISA Resource Name Control***



Interface object type: VISA
Vendor: ni
Resource name: TCPIP::10.85.0.68::INSTR

***Figure 10: Creating new Interface Object type VISA.***

After pressing ***OK***, new item under ***Interface Objects*** called ***VISA-TCPIP-10.85.0.68-inst0*** will be available (see Figure 13). Selecting it and pressing ***Connect*** button will allow for communication with the instrument:



VISA-TCPIP-0-10.85.0.68-inst0

Connection

Connection status to 10.85.0.68: **Disconnected**

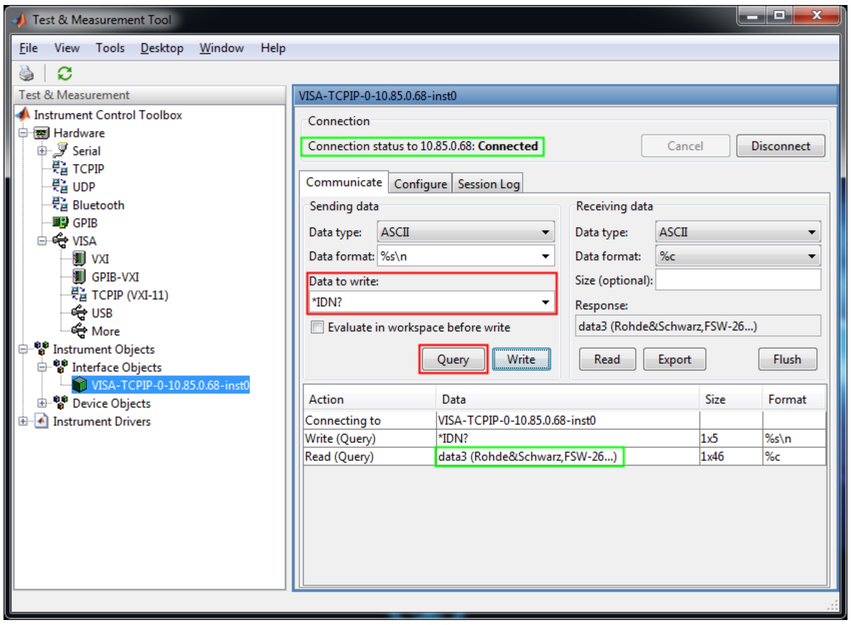
Sending data: Data type: ASCII; Data format: %s\n; Data to write: ; Evaluate in workspace before write

Receiving data: Data type: ASCII; Data format: %c; Size (optional): ; Response:

| Action | Data | Size | Format |
|---|---|---|---|

*Figure 11: Selection newly created Interface object and connecting to the instrument*

After connection to the instrument is successfully established, (see Figure 12) connection status changes to ***Connected*** and all controls are enabled. Use the field ***Data to write*** to send *IDN? string. Since this is a query (response from instrument is expected), press ***Query*** button. The response will be displayed in a log table:



VISA-TCPIP-0-10.85.0.68-inst0

Connection

Connection status to 10.85.0.68: **Connected**

Sending data: Data type: ASCII; Data format: %s\n; Data to write: *IDN?; Evaluate in workspace before write

Receiving data: Data type: ASCII; Data format: %c; Size (optional): ; Response: data3 (Rohde&Schwarz,FSW-26...)

| Action | Data | Size | Format |
|---|---|---|---|
| Connecting to | VISA-TCPIP-0-10.85.0.68-inst0 | | |
| Write (Query) | *IDN? | 1x5 | %s\n |
| Read (Query) | data3 (Rohde&Schwarz,FSW-26...) | 1x46 | %c |

*Figure 12: Communication with Interface Object VISA*

Besides being able to use interactive control of the instrument, switching to ***Session Log*** tab, reveals actual code that can be imported to other MATLAB scripts. In our case, this code looks like this (also available in attached files as raw_SCPI_communication.m):

```
% Find a VISA-TCPIP object.
obj1 = instrfind('Type', 'visa-tcpip', 'RsrcName',
'TCPIP0::10.85.0.68::inst0::INSTR', 'Tag', '');

% Create the VISA-TCPIP object if it does not exist
% otherwise use the object that was found.
if isempty(obj1)
    obj1 = visa('NI', 'TCPIP0::10.85.0.68::inst0::INSTR');
else
    fclose(obj1);
    obj1 = obj1(1)
end

% Connect to instrument object, obj1.
fopen(obj1);

% Communicating with instrument object, obj1.
data1 = query(obj1, '*IDN?');

% Disconnect from instrument object, obj1.
fclose(obj1);
```

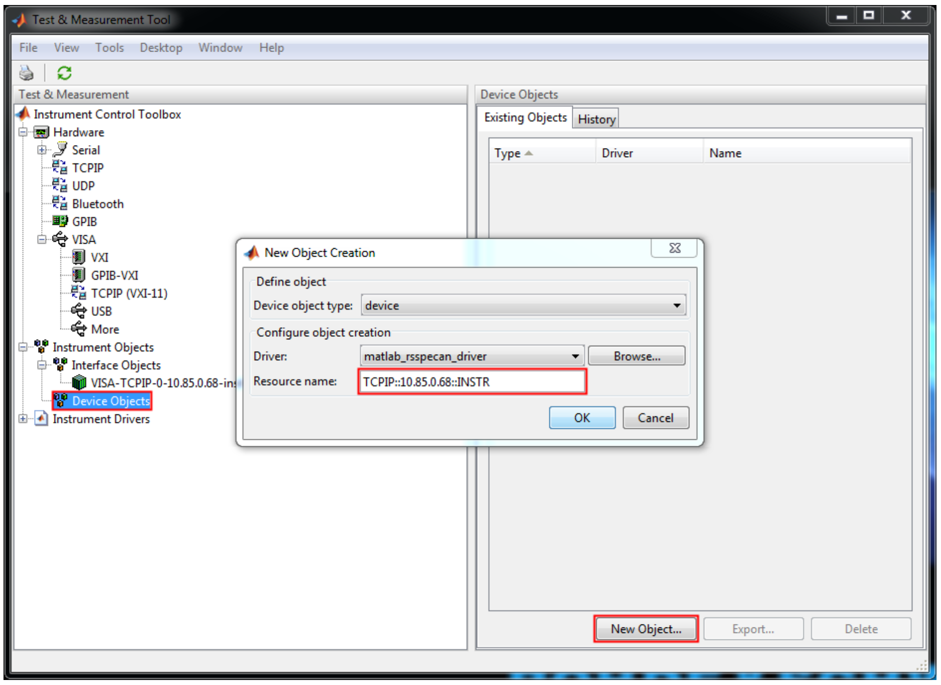
**Important:** Tab ***Configure*** contains settings for the object (`obj1` in our case). Input and output buffers are set by default to 512 bytes. If the commands or responses are expected to be longer, output / input buffer size should be change to appropriate values, otherwise commands / responses will be truncated. Changing any of the properties also generates MATLAB code in ***Session Log*** tab which can be used. In our case the following commands are generated:

```
% Configure instrument object, obj1
set(obj1, 'InputBufferSize', 2048);
set(obj1, 'OutputBufferSize', 4096);
```

## 2.6 Using MATLAB Instrument Driver Interactively

The target of this chapter is to interactively control instruments using the previously created instrument driver. The advantage of interactive control is a ***Session Log*** that can be just copied to MATLAB script. The result script at the end of this chapter will be from the functionality point of view identical with the method of direct programing explained in chapter 2.7.

In the TMTOOL left window select ***Instrument Objects -> Device Objects*** and press ***New Object****…* button (see Figure 13):



Define object
Device object type: device
Configure object creation
Driver: matlab_rsspecan_driver
Resource name: TCPIP::10.85.0.68::INSTR

***Figure 13: The first step is to click "New Object". Next, select the previously generated MATLAB instrument driver file (extension: .mdd) and enter Resource name.***

The previously generated MATLAB instrument driver file (extension: .mdd) file has to be selected as ***Driver***. This file was created in chapter 2.2.

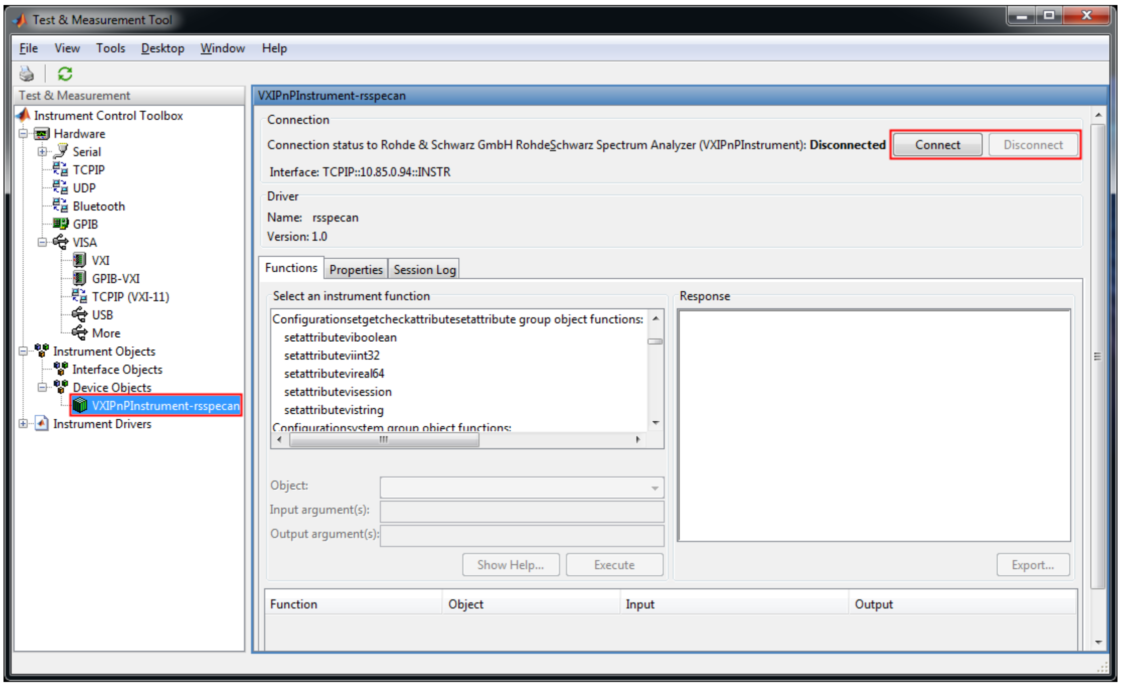
After configuring a valid ***VISA Resource name*** as ***Resource name*** (the same one is used in 2.5 explaining Raw SCPI communication) as new device object can be created by pressing the *OK* button.

In case the following error occurs (usually on 64-bit version of MATLAB):

**"A 'Selected' compiler was not found. You may need to run >> *mex -setup*"**

follow the instruction in 4.6.

Now, you can see the new item under ***Instrument Objects -> Device Objects*** *called* ***VXIPnPInstrument-rsspecan***. Selecting this item gives you the overview of all driver's functions and properties. You can also establish and close a connection with instrument using ***Connect / Disconnect*** buttons (see Figure 14):



Connection status to Rohde & Schwarz GmbH RohdeSchwarz Spectrum Analyzer (VXIPnPInstrument): **Disconnected**

Interface: TCPIP::10.85.0.94::INSTR

Name: rsspecan
Version: 1.0

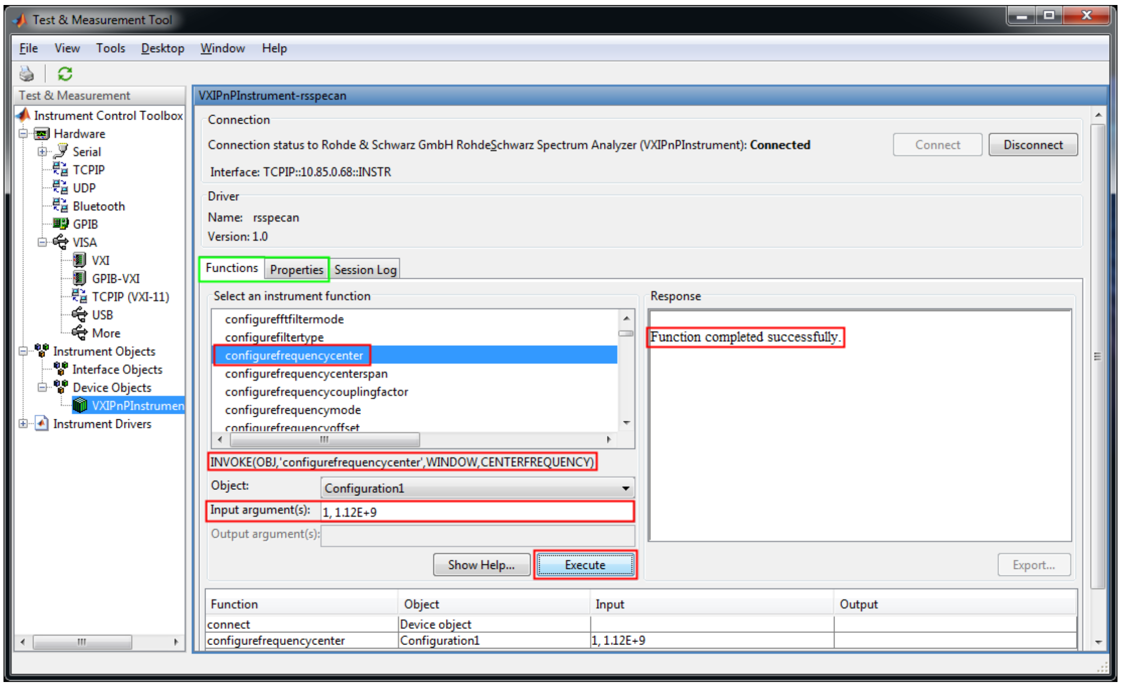
Configurationsetgetcheckattributesetattribute group object functions:
setattributeviboolean
setattributeviint32
setattributevireal64
setattributevisession
setattributevistring
Configurationsystem group object functions:

***Figure 14: The Functions tab allows you to select the functionality of the MATLAB instrument driver Use Connect / Disconnect buttons to initiate or end the connection.***

There are 2 main approaches to use the instrument driver: Functions and Attributes (in TMTOOL they are called Properties). Hence the name attribute-based drivers. Using the Properties is more universal and most of the Functions call Properties inside. However, reading the parameter values (for example actual spectrum analyzer center frequency) can be only achieved using the Properties. In the following two sub-chapters 2.6.1 and 2.6.2 setting of a spectrum analyzer center frequency is explained using both approaches. Keep in mind, that the most effective use of instrument drivers is combining these two approaches together.

## 2.6.1 Functions settings with TMTOOL

In this example we will interactively set the spectrum analyzer center frequency to 1.12GHz by using the high-level function.



Connection

Connection status to Rohde & Schwarz GmbH RohdeSchwarz Spectrum Analyzer (VXIPnPInstrument): **Connected**

Interface: TCPIP::10.85.0.68::INSTR

Driver

Name: rsspecan

Version: 1.0

INVOKE(OBJ,'configurefrequencycenter',WINDOW,CENTERFREQUENCY)

Object: Configuration1

Input argument(s): 1, 1.12E+9

Output argument(s):

Function completed successfully.

| Function | Object | Input | Output |
|---|---|---|---|
| connect | Device object | | |
| configurefrequencycenter | Configuration1 | 1, 1.12E+9 | |

***Figure 15: Setting center frequency by using Function configurefrequencycenter***

Press the ***Connect*** Button and switch to ***Functions*** tab (see Figure 15).

Scroll to the ***configurefrequencycenter*** function in ***Configuration*** group (faster and more convenient way is described in 2.7.1) and notice the function call:

```
INVOKE(OBJ,'configurefrequencycenter',WINDOW,CENTERFREQUENCY)
```

Fill the ***Input argument(s)*** field with the following value (corresponding parameters are highlighted with the same color:

```
1, 1.12E+9
```

Press ***Execute***. The ***Response*** field reports the status of the executed function.

Press ***Disconnect*** button and switch to ***Session Log*** tab.
There, you can now find the MATLAB script log of all that you have done and just copy and paste it your code. This script is also available in attached files as rsspecan_TMTOOL_functions_example.m.

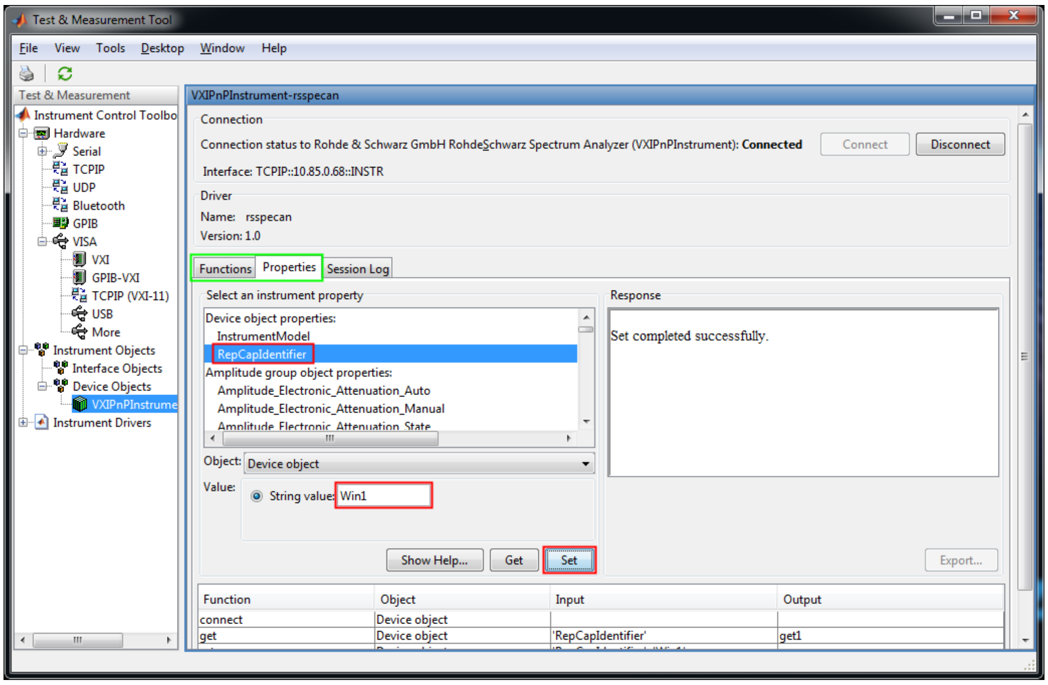
```
% Create a device object.
deviceObj = icdevice('matlab_rsspecan_driver.mdd',
'TCPIP::10.85.0.68::INSTR');
```

```
% Connect device object to hardware.
connect(deviceObj);

% Execute device object function(s).
groupObj = get(deviceObj, 'Configuration');
groupObj = groupObj(1);
invoke(groupObj, 'configurefrequencycenter', 1, 1.12E+9);

% Disconnect device object from hardware.
disconnect(deviceObj);
% Delete object
delete(deviceObj);
```

## 2.6.2 Properties settings with TMTOOL

In this example we will do the same operation as in the previous chapter – setting the center frequency to 1.12GHz, but we will be using the property setting instead. In addition, we will also read the same property value back.

Press the ***Connect*** Button and switch to ***Properties*** tab (see Figure 16):



Connection

Connection status to Rohde & Schwarz GmbH RohdeSchwarz Spectrum Analyzer (VXIPnPInstrument): **Connected**

Interface: TCPIP::10.85.0.68::INSTR

Driver

Name: rsspecan

Version: 1.0

Select an instrument property

Device object properties:
InstrumentModel
RepCapIdentifier
Amplitude group object properties:
Amplitude_Electronic_Attenuation_Auto
Amplitude_Electronic_Attenuation_Manual
Amplitude_Electronic_Attenuation_State

Object: Device object

Value: String value: Win1

Response

Set completed successfully.

| Function | Object | Input | Output |
|---|---|---|---|
| connect | Device object | | |
| get | Device object | 'RepCapIdentifier' | get1 |

*Figure 16: Setting the RepCapIdentifier property*

Select the ***RepCapIdentifier***, set the field ***Value*** to Win1 and press ***Set*** button. This sets the repeated capability to Window 1, which is the equivalent of the 1st parameter WINDOW of the function ***configurefrequencycenter***.

Scroll down to ***Basicoperation*** group and select ***Center_Frequency*** property (faster and more convenient way is described in 2.7.2). Set the field ***Value*** to 1.12E+9 and press ***Set*** button. Then, use ***Get*** button to read the response.

Press ***Disconnect*** button and switch to ***Session Log*** tab.
Generated script is also a part of rsspecan_TMTOOL_properties_example.m.

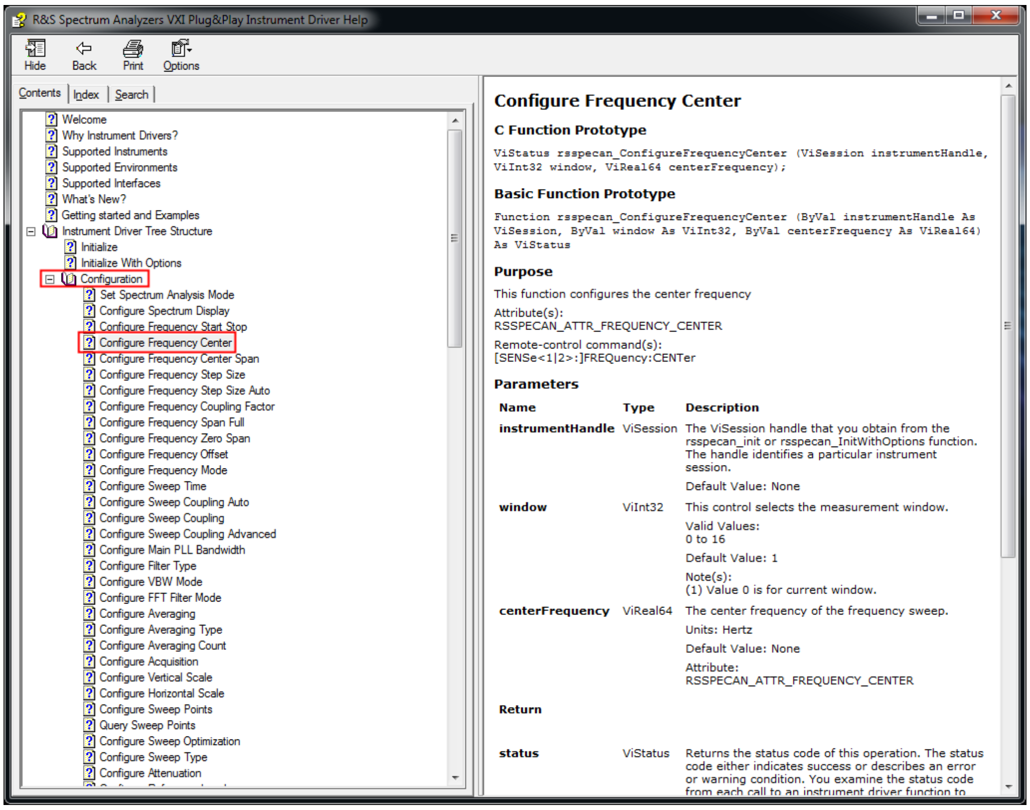
```
% Create a device object.
deviceObj = icdevice('matlab_rsspecan_driver.mdd',
'TCPIP::10.85.0.68::INSTR');

% Connect device object to hardware.
connect(deviceObj);

% Configure property value(s).
set(deviceObj, 'RepCapIdentifier', 'Win1');
set(deviceObj.Basicoperation(1), 'Center_Frequency', 1.12E+9);

% Query property value(s).
get2 = get(deviceObj.Basicoperation(1), 'Center_Frequency');

% Disconnect device object from hardware.
disconnect(deviceObj);
% Delete object
delete(deviceObj);
```

## 2.7 Composing the remote-control MATLAB scripts directly

After a little experience you will find that creating instrument remote control MATLAB script is more efficient just by using Instrument Driver Editor and Rohde & Schwarz VXIplug&play help files.
This chapter shows how you can achieve the same results as described in previous two chapters i.e. composing the MATLAB remote-control script faster.

### 2.7.1 Functions settings with Instrument Driver Editor

Open Rohde & Schwarz Instrument driver help file described in 2.4.1 and navigate to ***Instrument Driver Tree Structure -> Configuration -> Configure Frequency Center*** (see Figure 17):


R&S Spectrum Analyzers VXI Plug&Play Instrument Driver Help
Hide
Back
Print
Options
Contents
Index
Search
Welcome
Why Instrument Drivers?
Supported Instruments
Supported Environments
Supported Interfaces
What's New?
Getting started and Examples
Instrument Driver Tree Structure
Initialize
Initialize With Options
Configuration
Set Spectrum Analysis Mode
Configure Spectrum Display
Configure Frequency Start Stop
Configure Frequency Center
Configure Frequency Center Span
Configure Frequency Step Size
Configure Frequency Step Size Auto
Configure Frequency Coupling Factor
Configure Frequency Span Full
Configure Frequency Zero Span
Configure Frequency Offset
Configure Frequency Mode
Configure Sweep Time
Configure Sweep Coupling Auto
Configure Sweep Coupling
Configure Sweep Coupling Advanced
Configure Main PLL Bandwidth
Configure Filter Type
Configure VBW Mode
Configure FFT Filter Mode
Configure Averaging
Configure Averaging Type
Configure Averaging Count
Configure Acquisition
Configure Vertical Scale
Configure Horizontal Scale
Configure Sweep Points
Query Sweep Points
Configure Sweep Optimization
Configure Sweep Type
Configure Attenuation

**Configure Frequency Center**

**C Function Prototype**

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

**Basic Function Prototype**

```
Function rsspecan_ConfigureFrequencyCenter (ByVal instrumentHandle As
ViSession, ByVal window As ViInt32, ByVal centerFrequency As ViReal64)
As ViStatus
```

**Purpose**

This function configures the center frequency

Attribute(s):
RSSPECAN_ATTR_FREQUENCY_CENTER

Remote-control command(s):
[SENSe<1|2>:]FREQuency:CENTer

**Parameters**

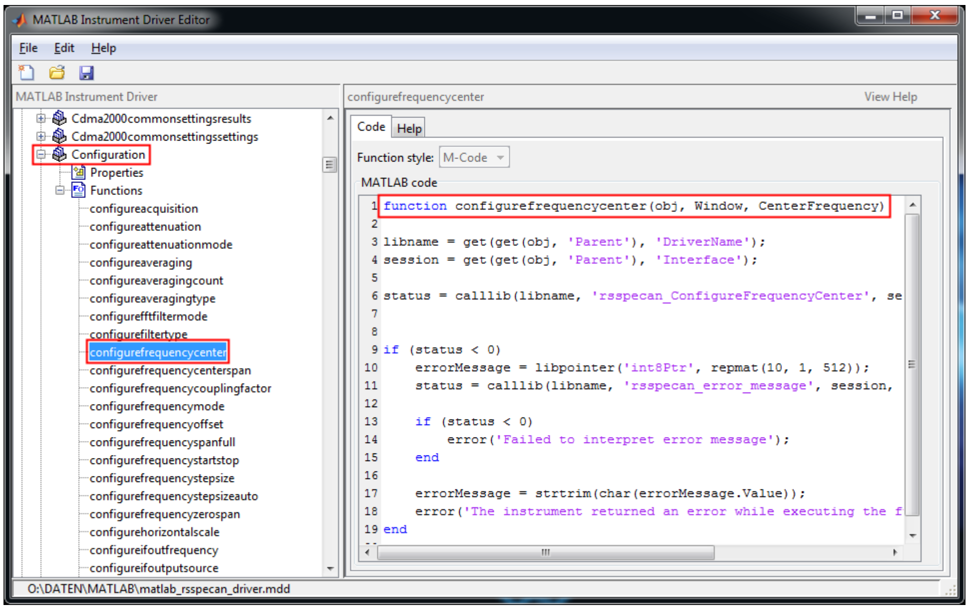
| Name | Type | Description |
|---|---|---|
| **instrumentHandle** | ViSession | The ViSession handle that you obtain from the rsspecan_init or rsspecan_InitWithOptions function. The handle identifies a particular instrument session. Default Value: None |
| **window** | ViInt32 | This control selects the measurement window. Valid Values: 0 to 16 Default Value: 1 Note(s): (1) Value 0 is for current window. |
| **centerFrequency** | ViReal64 | The center frequency of the frequency sweep. Units: Hertz Default Value: None Attribute: RSSPECAN_ATTR_FREQUENCY_CENTER |

**Return**

| | | |
|---|---|---|
| **status** | ViStatus | Returns the status code of this operation. The status code either indicates success or describes an error or warning condition. You examine the status code from each call to an instrument driver function to |

*Figure 17: Rohde & Schwarz Instrument Driver help file for function rsspecan_ConfigureFrequencyCenter in group Configuration.*

Open MATLAB Instrument Driver Editor for ***matlab_rsspecan_driver*** (described in 2.3) and search for the same function (Figure 18):



```
Cdma2000commonsettingsresults
Cdma2000commonsettingssettings
Configuration
    Properties
    Functions
        configureacquisition
        configureattenuation
        configureattenuationmode
        configureaveraging
        configureaveragingcount
        configureaveragingtype
        configurefftfiltermode
        configurefiltertype
        configurefrequencycenter
        configurefrequencycenterspan
        configurefrequencycouplingfactor
        configurefrequencymode
        configurefrequencyoffset
        configurefrequencyspanfull
        configurefrequencystartstop
        configurefrequencystepsize
        configurefrequencystepsizeauto
        configurefrequencyzerospan
        configurehorizontalscale
        configureifoutfrequency
        configureifoutputsource
```

Function style: M-Code

```
function configurefrequencycenter(obj, Window, CenterFrequency)

libname = get(get(obj, 'Parent'), 'DriverName');
session = get(get(obj, 'Parent'), 'Interface');

status = calllib(libname, 'rsspecan_ConfigureFrequencyCenter', se


if (status < 0)
    errorMessage = libpointer('int8Ptr', repmat(10, 1, 512));
    status = calllib(libname, 'rsspecan_error_message', session,

    if (status < 0)
        error('Failed to interpret error message');
    end

    errorMessage = strtrim(char(errorMessage.Value));
    error('The instrument returned an error while executing the f
end
```

O:\DATEN\MATLAB\matlab_rsspecan_driver.mdd

***Figure 18: MATLAB Instrument Driver Editor with selected function group Configuration and function configurefrequencycenter prototype***

As you can see, the *makemid* tool removes the prefix of VXIplug&play function names. Furthermore all capital letters of the VXIplug&play function name are lowercased as well, for example ***rsspecan_ConfigureFrequencyCenter (…)*** will be converted to ***configurefrequencycenter***. The tree structure is flattened to 2 levels where the 2nd level is the function name and 1st level is all previous level names combined – this is called the function group. Also, all functions and groups are alphabetically ordered.

You can also notice missing INIT and CLOSE functions. That is because those functions mandatory for VXIplug&play drivers and are called by MATLAB instrument driver functions `connect()` and `disconnect()`.

In addition, function name is not sufficient to call it, the function group has to be provided as well. In our example function group is called ***Configuration*** (see Figure 18).

---

Remote control of the instrument starts with creating device object using previously created MATLAB instrument driver (described in 2.2) with defined VISA Resource name:

```
deviceObj = icdevice('matlab_rsspecan_driver.mdd',
'TCPIP::10.85.0.68::INSTR');
```

---

Then, establish the connection to the hardware:

```
% Connect device object to hardware.
connect(deviceObj);
```

---

For setting the center frequency, navigate in MATLAB Instrument Driver Editor to group ***Configuration*** and function ***configurefrequencycenter.*** The fastest way is to use the Rohde & Schwarz Instrument driver help file with search function and then following the same path in MATLAB Instrument Driver Editor. However, the order of functions and groups is not kept the same, but sorted alphabetically.
Notice the function prototype in MATLAB Instrument Driver Editor ***Code*** tab window in Figure 18:

```
function configurefrequencycenter(obj, Window, CenterFrequency)
```

Our code for setting the center frequency will look like this (corresponding parameters are highlighted with the same colors):

```
% Creating Group object.
groupObj = get(deviceObj, 'Configuration');
groupObj = groupObj(1);
% Execute device object function.
invoke(groupObj, 'configurefrequencycenter', 1, 1.12E+9);
```

We're using function `invoke` with group object `groupObj`, function name and then all parameters in the same order except the 1st one (`obj`).
Here's the dense version of the code that is used in all the following examples:

```
invoke(get(deviceObj, 'Configuration'),
'configurefrequencycenter', 1, 1.12E+9);
```

You can see there are no return parameters. Error code returned by VXIplug&play driver function is handled in MATLAB instrument driver and you need to use the `try` construct to handle it:

```
try
  try block...
catch
  catch block...
end
```

Default behavior is stopping the program if an error occurs. See the complete script example rsspecan_TMTOOL_complex_example.m in attachment using this construct.

For comparison, example rsspecan_calllib_complex_example_32bit.m has to use the following line after each calllib call to handle errors:

```
if (err) break;  end
```

---

Writing and reading data using `queryvistring()` function following the same steps:

Function prototype in MATLAB Instrument Driver Editor ***Code*** tab window:

```
function [Value] = queryvistring(obj, Command, BufferSize)
```

MATLAB code:

```
[Value] = invoke(get(deviceObj, 'Utilityfunctionsinstrumentio'),
'queryvistring', '*IDN?', 200);
```

The variable `Value` will then contain the response from the instrument.

---

Next task will be setting analyzer to single sweep, performing the sweep and reading the trace afterwards:

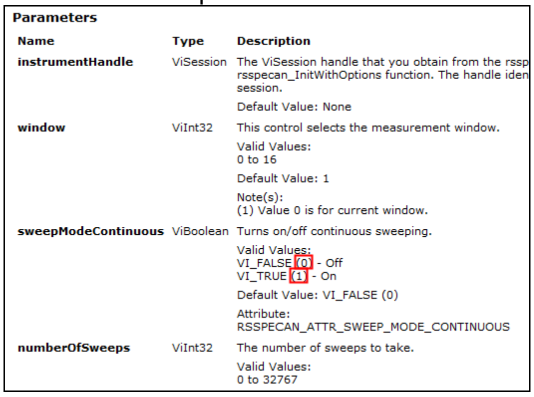
Navigate to group ***Configuration***, function ***configureacquistion.*** Function prototype:

```
function configureacquisition(obj, Window, SweepModeContinuous,
NumberOfSweeps)
```

MATLAB code:

```
invoke(get(deviceObj, 'Configuration'), 'configureacquisition',
1, 0, 1);
```

Please note that you cannot use the defined constants e.g. VI_FALSE / VI_TRUE mentioned in the driver help. This is because MATLAB  cannot include existing header files. You must use their numeric interpretations mentioned in the brackets:



**Parameters**

| Name | Type | Description |
|---|---|---|
| **instrumentHandle** | ViSession | The ViSession handle that you obtain from the rssp rsspecan_InitWithOptions function. The handle iden session.<br>Default Value: None |
| **window** | ViInt32 | This control selects the measurement window.<br>Valid Values:<br>0 to 16<br>Default Value: 1<br>Note(s):<br>(1) Value 0 is for current window. |
| **sweepModeContinuous** | ViBoolean | Turns on/off continuous sweeping.<br>Valid Values:<br>VI_FALSE $\boxed{(0)}$ - Off<br>VI_TRUE $\boxed{(1)}$ - On<br>Default Value: VI_FALSE (0)<br>Attribute:<br>RSSPECAN_ATTR_SWEEP_MODE_CONTINUOUS |
| **numberOfSweeps** | ViInt32 | The number of sweeps to take.<br>Valid Values:<br>0 to 32767 |

***Figure 19: ConfigureAcquisition VXIpnp Instrument driver help***

Navigate to group ***Measurementlowlevelmeasurement***, function ***initiate***:

```
function initiate(obj, Window, Timeout)
```

MATLAB code:

```
invoke(get(deviceObj, 'Measurementlowlevelmeasurement'),
'initiate', 1, 5000);
```

Navigate to group ***Configuration***, function ***querysweeppoints***:

```
function [SweepPoints] = querysweeppoints(obj, Window)
```

MATLAB code:

```
[ArrayLength] = invoke(get(deviceObj, 'Configuration'),
'QuerySweepPoints', 1);
```

Navigate to group ***Measurement***, function ***fetchytrace.*** Function prototype:

```
function [ActualPoints, Amplitude] = fetchytrace(obj, Window,
Trace, ArrayLength, Amplitude)
```

MATLAB code:

```
Amplitude = zeros(ArrayLength, 1); %initialize array with zeroes
[ActualPoints, Amplitude] = invoke(get(deviceObj,
'Measurement'), 'fetchytrace', 1, 1, ArrayLength, Amplitude);
```

Notice, that the variable `Amplitude` is mentioned in both input and output parameters. That is because it is an array of numbers and it needs an allocated buffer before the function `fetchytrace` is invoked. Compared with scalar return value of the function `querysweeppoints` the `SweepPoints` parameter is only mentioned in output parameters, because the function only returns one (scalar) float value whose size is known upfront.

---

Finally, end the connection and delete the Device Object:

```
% Disconnect device object to hardware.
disconnect(deviceObj);
% Delete the  device object.
delete(deviceObj);
```

Deleting the device object prevents multiple objects to be created every time you call `icdevice` function:

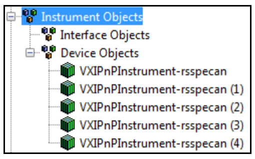
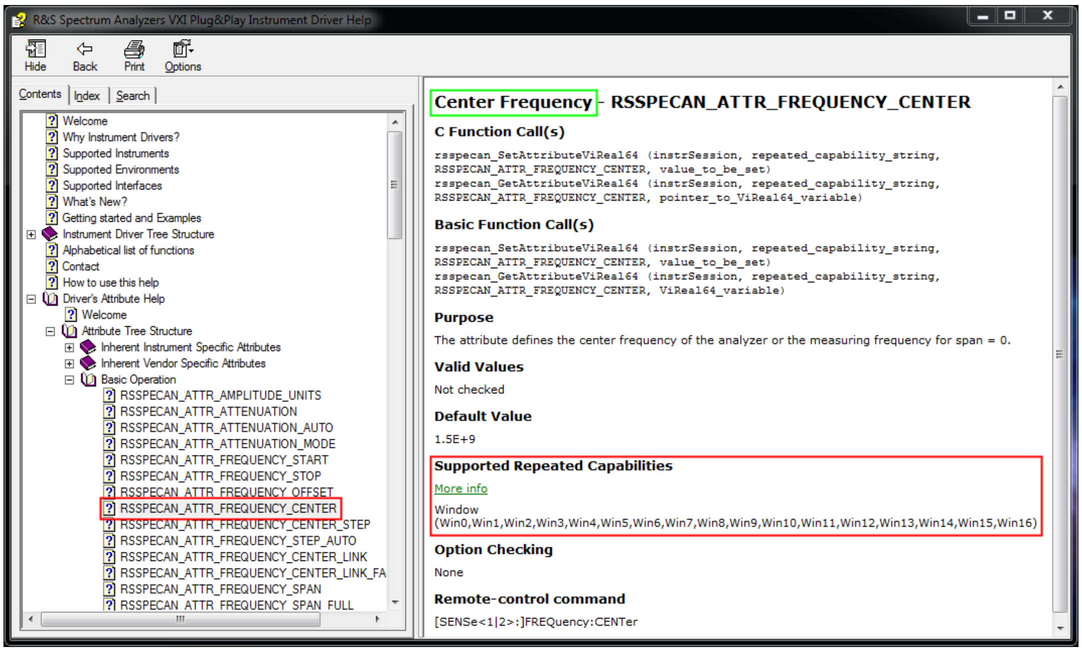

Instrument Objects
Interface Objects
Device Objects
VXIPnPInstrument-rsspecan
VXIPnPInstrument-rsspecan (1)
VXIPnPInstrument-rsspecan (2)
VXIPnPInstrument-rsspecan (3)
VXIPnPInstrument-rsspecan (4)

***Figure 20: Multiple rsspecan Device Objects in TMTOOL***

The entire script is in attached files as
rsspecan_TMTOOL_direct_functions_example.m

### 2.7.2 Properties settings with Instrument Driver Editor

Open Rohde & Schwarz Instrument driver help file described in 2.4.1 and navigate to ***Driver's Attribute Help -> Attribute tree structure -> Basic Operation -> RSSPECAN_ATTR_FREQUENCY_CENTER***



Welcome
Why Instrument Drivers?
Supported Instruments
Supported Environments
Supported Interfaces
What's New?
Getting started and Examples
Instrument Driver Tree Structure
Alphabetical list of functions
Contact
How to use this help
Driver's Attribute Help
  Welcome
  Attribute Tree Structure
    Inherent Instrument Specific Attributes
    Inherent Vendor Specific Attributes
    Basic Operation
      RSSPECAN_ATTR_AMPLITUDE_UNITS
      RSSPECAN_ATTR_ATTENUATION
      RSSPECAN_ATTR_ATTENUATION_AUTO
      RSSPECAN_ATTR_ATTENUATION_MODE
      RSSPECAN_ATTR_FREQUENCY_START
      RSSPECAN_ATTR_FREQUENCY_STOP
      RSSPECAN_ATTR_FREQUENCY_OFFSET
      RSSPECAN_ATTR_FREQUENCY_CENTER
      RSSPECAN_ATTR_FREQUENCY_CENTER_STEP
      RSSPECAN_ATTR_FREQUENCY_STEP_AUTO
      RSSPECAN_ATTR_FREQUENCY_CENTER_LINK
      RSSPECAN_ATTR_FREQUENCY_CENTER_LINK_FA
      RSSPECAN_ATTR_FREQUENCY_SPAN
      RSSPECAN_ATTR_FREQUENCY_SPAN_FULL

**Center Frequency - RSSPECAN_ATTR_FREQUENCY_CENTER**

**C Function Call(s)**

```
rsspecan_SetAttributeViReal64 (instrSession, repeated_capability_string,
RSSPECAN_ATTR_FREQUENCY_CENTER, value_to_be_set)
rsspecan_GetAttributeViReal64 (instrSession, repeated_capability_string,
RSSPECAN_ATTR_FREQUENCY_CENTER, pointer_to_ViReal64_variable)
```

**Basic Function Call(s)**

```
rsspecan_SetAttributeViReal64 (instrSession, repeated_capability_string,
RSSPECAN_ATTR_FREQUENCY_CENTER, value_to_be_set)
rsspecan_GetAttributeViReal64 (instrSession, repeated_capability_string,
RSSPECAN_ATTR_FREQUENCY_CENTER, ViReal64_variable)
```

**Purpose**

The attribute defines the center frequency of the analyzer or the measuring frequency for span = 0.

**Valid Values**

Not checked

**Default Value**

1.5E+9

**Supported Repeated Capabilities**

More info

Window
(Win0,Win1,Win2,Win3,Win4,Win5,Win6,Win7,Win8,Win9,Win10,Win11,Win12,Win13,Win14,Win15,Win16)

**Option Checking**

None

**Remote-control command**

[SENSe<1|2>:]FREQuency:CENTer

***Figure 21: rsspecan instrument driver help property RSSPECAN_ATTR_FREQUENCY_CENTER***

Notice the name "**Center Frequency**" (green square) and Supported Repeated capabilities e.g. `'Win1'`

Take the Property name from Figure 21 green rectangle; replace spaces with underscore characters ('_'). In our case this will result in property name ***Center_Frequency***. Now we have to find out group name:

Group name as described in 2.7.1 is the combined text of all directory levels above, without spaces or dashes, starting after ***Attribute tree structure***. The entire name is lower-cased except the 1st character.

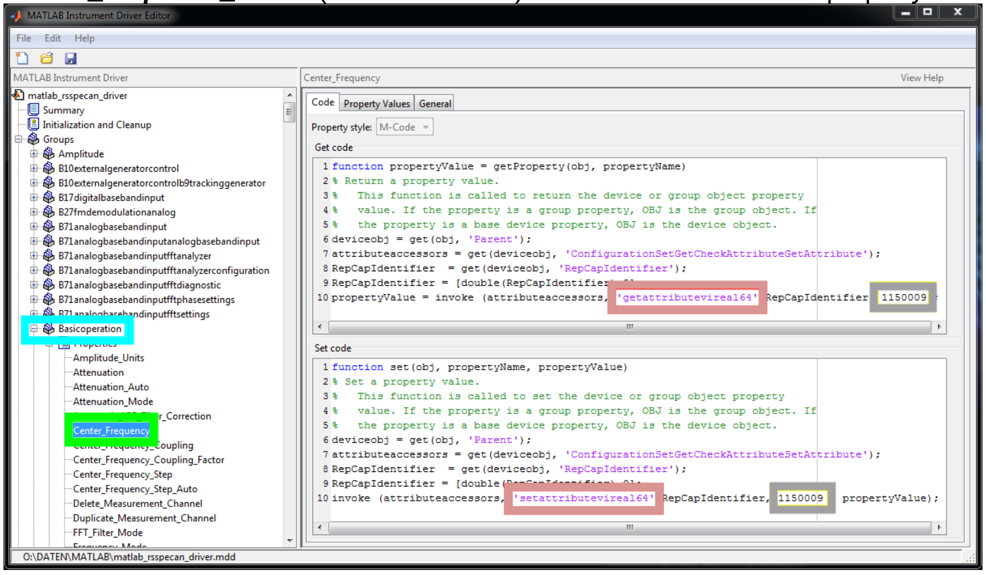
In our example:
***Driver's Attribute Help -> Attribute tree structure -> Basic Operation***

the group name will be ***Basicoperation***

If there are more levels, like:
***Driver's Attribute Help -> Attribute tree structure -> Basic Operation -> Low-Level Measurement***

The group name will be ***Basicoperationlowlevelmeasurement***

If you are in doubt, open MATLAB Instrument Driver Editor for ***matlab_rsspecan_driver*** (described in 2.3) and search for the same property:



Property style: M-Code

Get code

```
function propertyValue = getProperty(obj, propertyName)
% Return a property value.
%   This function is called to return the device or group object property
%   value. If the property is a group property, OBJ is the group object. If
%   the property is a base device property, OBJ is the device object.
deviceobj = get(obj, 'Parent');
attributeaccessors = get(deviceobj, 'ConfigurationSetGetCheckAttributeGetAttribute');
RepCapIdentifier  = get(deviceobj, 'RepCapIdentifier');
RepCapIdentifier = [double(RepCapIdentifier
propertyValue = invoke (attributeaccessors, 'getattributevireal64' RepCapIdentifier 1150009
```

Set code

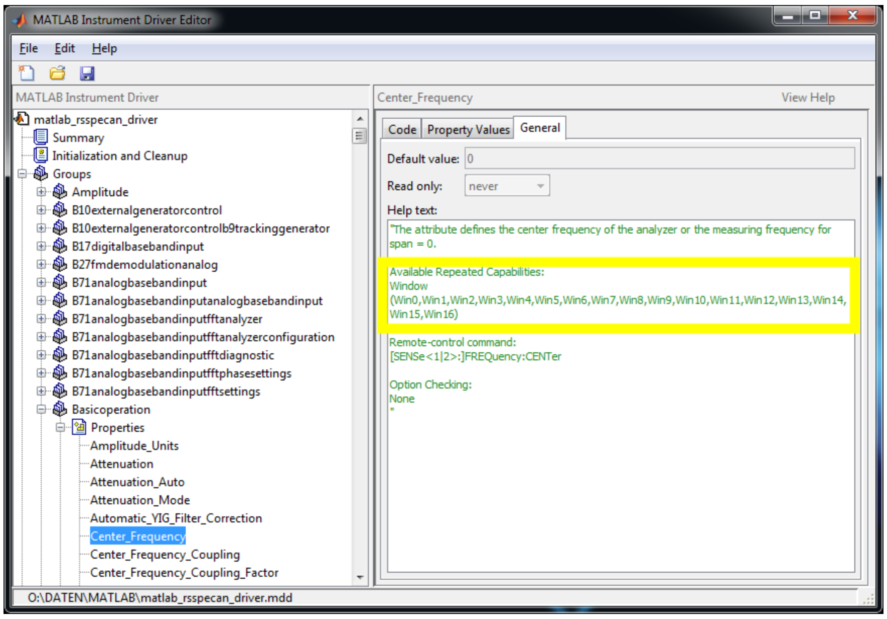
```
function set(obj, propertyName, propertyValue)
% Set a property value.
%   This function is called to set the device or group object property
%   value. If the property is a group property, OBJ is the group object. If
%   the property is a base device property, OBJ is the device object.
deviceobj = get(obj, 'Parent');
attributeaccessors = get(deviceobj, 'ConfigurationSetGetCheckAttributeSetAttribute');
RepCapIdentifier  = get(deviceobj, 'RepCapIdentifier');
RepCapIdentifier = [double(
invoke (attributeaccessors, 'setattributevireal64' RepCapIdentifier, 1150009  propertyValue);
```

O:\DATEN\MATLAB\matlab_rsspecan_driver.mdd

***Figure 22: MATLAB  Instrument Driver Editor property Center_Frequency. Frame colors correspond to highlighted parameters in MATLAB  script.***

There, you can easily see the group name and also the property name.
In addition, there is a direct way to set/get property values. For that you need the appropriate function depending on property value type (…`vireal64` or …`viint32` …`viboolean` or …`vistring`). This can be copied from ***Code*** tab ***Set code*** and ***Get code*** fields. You also need property ID number. See the MATLAB script code at the end of this chapter where the highlighted parts of code correspond color-wise to Figure 22 and Figure 23.

If you switch to ***General*** tab (Figure 23), you can find the same help that it is provided in Figure 21:


MATLAB Instrument Driver Editor
File Edit Help
MATLAB Instrument Driver
matlab_rsspecan_driver
Summary
Initialization and Cleanup
Groups
Amplitude
B10externalgeneratorcontrol
B10externalgeneratorcontrolb9trackinggenerator
B17digitalbasebandinput
B27fmdemodulationanalog
B71analogbasebandinput
B71analogbasebandinputanalogbasebandinput
B71analogbasebandinputfftanalyzer
B71analogbasebandinputfftanalyzerconfiguration
B71analogbasebandinputfftdiagnostic
B71analogbasebandinputfftphasesettings
B71analogbasebandinputfftsettings
Basicoperation
Properties
Amplitude_Units
Attenuation
Attenuation_Auto
Attenuation_Mode
Automatic_YIG_Filter_Correction
Center_Frequency
Center_Frequency_Coupling
Center_Frequency_Coupling_Factor
Center_Frequency
View Help
Code
Property Values
General

Default value: 0

Read only: never

Help text:

"The attribute defines the center frequency of the analyzer or the measuring frequency for span = 0.

Available Repeated Capabilities:
Window
(Win0,Win1,Win2,Win3,Win4,Win5,Win6,Win7,Win8,Win9,Win10,Win11,Win12,Win13,Win14,Win15,Win16)

Remote-control command:
[SENSe<1|2>:]FREQuency:CENTer

Option Checking:
None
"

O:\DATEN\MATLAB\matlab_rsspecan_driver.mdd

***Figure 23: MATLAB  Instrument Driver Editor Help for Center_Frequency property. Yellow frame color corresponds to highlighted parameter RepCapIdentifier in MATLAB  script.***

After finding out allowed repeated capability, property name and group name, we can compose the MATLAB code (also available in attached files as rsspecan_TMTOOL_properties_example.m):

```
% Create a device object.
deviceObj = icdevice('matlab_rsspecan_driver.mdd',
'TCPIP::10.85.0.68::INSTR');

% Connect device object to hardware.
connect(deviceObj);

% Configure Repeated Capability
set(deviceObj, 'RepCapIdentifier', 'Win1');

% Configure property value(s).
set(deviceObj.Basicoperation, 'Center_Frequency', 1.12E+9);

% Query property value(s).
Value = get(deviceObj.Basicoperation, 'Center_Frequency');

% Alternative way to set the property.
SetPropObj = get(deviceObj,
'ConfigurationSetGetCheckAttributeSetAttribute');
invoke (SetPropObj, 'setattributevireal64', 'Win1', 1150009 ,
1.12E+9);

% Alternative way to get the property.
GetPropObj = get(deviceObj,
'ConfigurationSetGetCheckAttributeGetAttribute');
Value = invoke (GetPropObj, 'getattributevireal64', 'Win1',
1150009 , 1.12E+9);

% Disconnect device object from hardware.
disconnect(deviceObj);
% Delete object
delete(deviceObj);
```

## 2.8 Complete application example using TMTOOL MATLAB Instrument Driver for rsspecan

This example is available in attached files as rsspecan_TMTOOL_complex_example.m

## 2.9 Known Problems

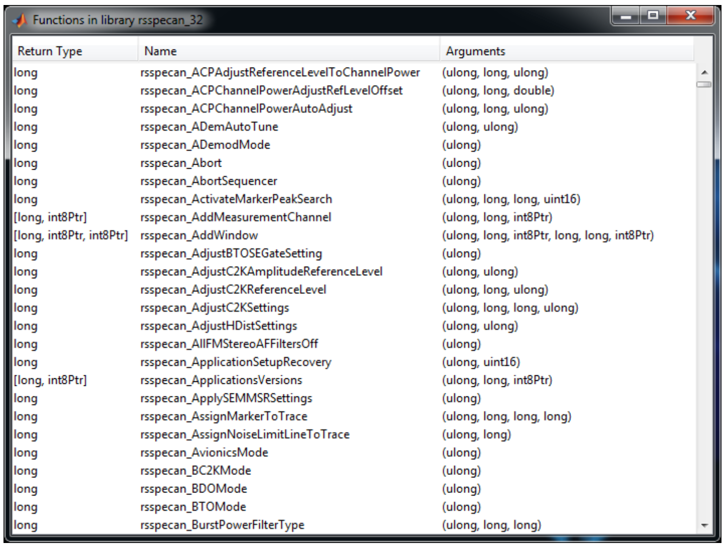
If the connection to the instrument driver doesn't work properly:

- Check you're the connection to the instrument in other independent program (NI MAX) or ***R&S Forum***.

- If MATLAB crashes during opening of the session (`connect`): For 32-bit version of MATLAB make sure you have the MATLAB Lcc-win32 compiler selected. Run:
  ```
  >> mex -setup
  ```
  and follow the instructions. Usually the Lcc-win32 is the choice nr. 1. Restart MATLAB

- Make sure you've installed the proper VXIplug&play instrument driver – for MATLAB 32-bit only the 32-bit driver can be used, for MATLAB 64-bit only the 64-bit driver can be used.

- Run the attached script rsspecan_MatlabCalllib_32bit.m or rsspecan_MatlabCalllib_64bit.m depending on your MATLAB version. The library must be loaded correctly. Otherwise the instrument driver cannot be properly used. The result of this script in case of successful execution is the window with library functions list:



| Return Type | Name | Arguments |
|---|---|---|
| long | rsspecan_ACPAdjustReferenceLevelToChannelPower | (ulong, long, ulong) |
| long | rsspecan_ACPChannelPowerAdjustRefLevelOffset | (ulong, long, double) |
| long | rsspecan_ACPChannelPowerAutoAdjust | (ulong, long, ulong) |
| long | rsspecan_ADemAutoTune | (ulong, ulong) |
| long | rsspecan_ADemodMode | (ulong) |
| long | rsspecan_Abort | (ulong) |
| long | rsspecan_AbortSequencer | (ulong) |
| long | rsspecan_ActivateMarkerPeakSearch | (ulong, long, long, uint16) |
| [long, int8Ptr] | rsspecan_AddMeasurementChannel | (ulong, long, int8Ptr) |
| [long, int8Ptr, int8Ptr] | rsspecan_AddWindow | (ulong, long, int8Ptr, long, long, int8Ptr) |
| long | rsspecan_AdjustBTOSEGateSetting | (ulong) |
| long | rsspecan_AdjustC2KAmplitudeReferenceLevel | (ulong, ulong) |
| long | rsspecan_AdjustC2KReferenceLevel | (ulong, long, ulong) |
| long | rsspecan_AdjustC2KSettings | (ulong, long, long, ulong) |
| long | rsspecan_AdjustHDistSettings | (ulong, ulong) |
| long | rsspecan_AllFMStereoAFFiltersOff | (ulong) |
| long | rsspecan_ApplicationSetupRecovery | (ulong, uint16) |
| [long, int8Ptr] | rsspecan_ApplicationsVersions | (ulong, long, int8Ptr) |
| long | rsspecan_ApplySEMMSRSettings | (ulong) |
| long | rsspecan_AssignMarkerToTrace | (ulong, long, long, long) |
| long | rsspecan_AssignNoiseLimitLineToTrace | (ulong, long) |
| long | rsspecan_AvionicsMode | (ulong) |
| long | rsspecan_BC2KMode | (ulong) |
| long | rsspecan_BDOMode | (ulong) |
| long | rsspecan_BTOMode | (ulong) |
| long | rsspecan_BurstPowerFilterType | (ulong, long, long) |

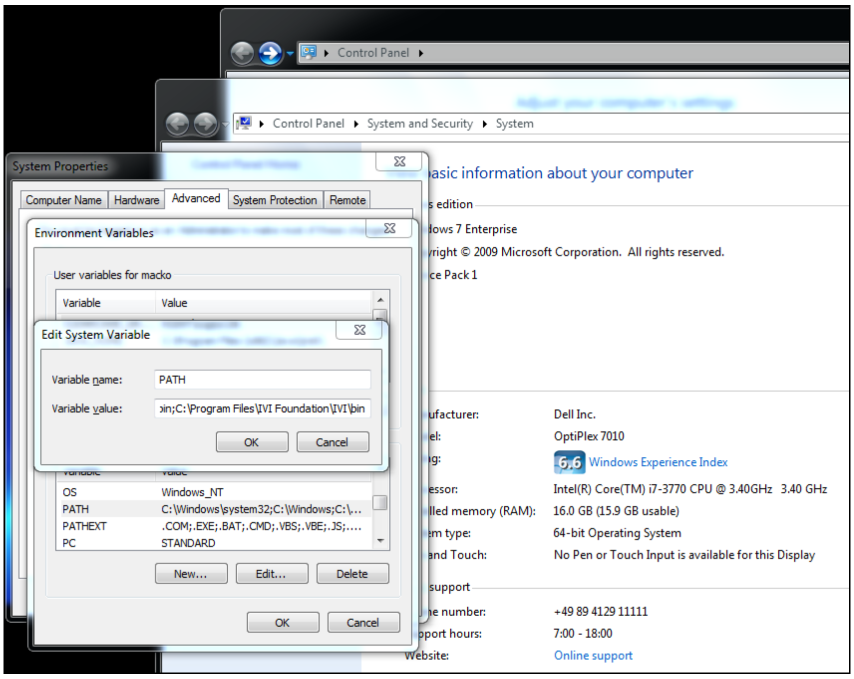
*Figure 24: Loaded library rsspecan_32.dll with list of all functions*

In case of an error, check environment variable ***PATH***:

```
>> getenv 'path'
```

Because of the MATLAB parsing process, none of the paths shall contain character '&', so for example ***C:\Program Files\Rohde&Schwarz*** will cause the inaccessibility of instrument driver dll. In this case:

Close MATLAB . Open system properties, tab Advanced, and Edit System Variable PATH (see Figure 25), delete the problematic path element or enclose the element with " " e.g. ***"C:\Program Files\Rohde&Schwarz"***


Control Panel ▸

Control Panel ▸ System and Security ▸ System

System Properties

Computer Name | Hardware | Advanced | System Protection | Remote

Environment Variables

User variables for macko

| Variable | Value |
|---|---|

Edit System Variable

Variable name: PATH

Variable value: ›in;C:\Program Files\IVI Foundation\IVI\bin

OK Cancel

| Variable | Value |
|---|---|
| OS | Windows_NT |
| PATH | C:\Windows\system32;C:\Windows;C:\... |
| PATHEXT | .COM;.EXE;.BAT;.CMD;.VBS;.VBE;.JS;.... |
| PC | STANDARD |

New... Edit... Delete

OK Cancel

asic information about your computer

s edition

lows 7 Enterprise

yright © 2009 Microsoft Corporation. All rights reserved.

ce Pack 1

| | |
|---|---|
| ufacturer: | Dell Inc. |
| el: | OptiPlex 7010 |
| ng: | 6,6 Windows Experience Index |
| essor: | Intel(R) Core(TM) i7-3770 CPU @ 3.40GHz 3.40 GHz |
| lled memory (RAM): | 16.0 GB (15.9 GB usable) |
| em type: | 64-bit Operating System |
| and Touch: | No Pen or Touch Input is available for this Display |

support

| | |
|---|---|
| e number: | +49 89 4129 11111 |
| pport hours: | 7:00 - 18:00 |
| Website: | Online support |

*Figure 25: Editing System Variable PATH in Windows 7*

# 3 Working with R&S custom made drivers

As you can see from previous chapters calling the instrument driver function means composing the `invoke` command and finding the group object and the function name. The reason for this approach is that MATLAB instrument driver is universal also for object – oriented drivers (IVI.COM or IVI.NET) in which the method name is not unique. There, the group object specifies the objects path to the method. However, in VXI plug&play driver this is not necessary and the function name is always unique. The same is also true for properties (attributes).

Therefore, Rohde & Schwarz provides custom made MATLAB driver (mdd driver, since it has the ".mdd " extension). You can find it in the same directory as **rsspecan_vxi.chm** help files: 32-bit VXI plug&play driver: **c:\Program Files (x86)\IVI Foundation\VISA\WinNT\rsspecan,** 64-bit VXI plug&play driver: **c:\Program Files\IVI Foundation\VISA\Win64\rsspecan**

It makes no difference whether it's the 32-bit or 64-bit version of the driver; mdd driver is always the same. Use the contact information in 7 to ask for the specific mdd driver if it is not included.

Rohde & Schwarz custom made mdd driver doesn't use any `groupObj`, but only the `deviceObj` created at the beginning with `icdevice`. What that means in terms of using functions and properties is described in the following chapters. For easier start, ready-to-use scripts using custom mdd drivers are attached: for rsspecan CUSTOM_rsspecan_TMTOOL_Complex_example.m and rsnrpz CUSTOM_rsnrpz_TMTOOL_example.m.

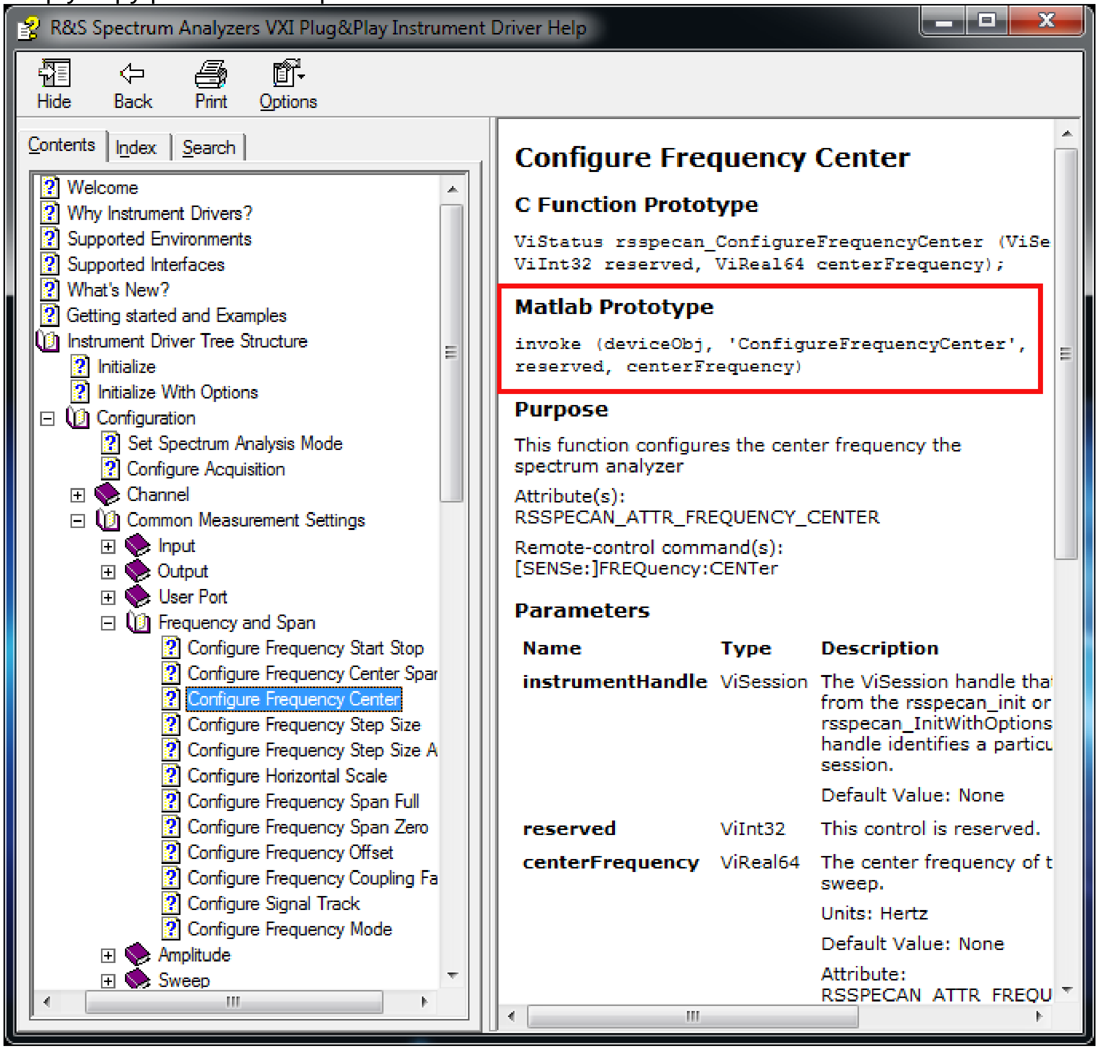
## 3.1 Calling functions

Composing any function call script always looks similar to this:

```
invoke(deviceObj, 'ConfigureFrequencyCenterSpan', SPAwindow,
SPAfrequencyCenter, SPAfrequencySpan);
```

Notice the function name: it is literally taken from original function name `'rsspecan_ConfigureFrequencyCenterSpan'` , just without the prefix. Therefore you can use ***rsspecan_vxi.chm*** for copy and paste. There, you also have the description of all parameters and their possible values.

New versions of R&S instrument drivers (rsspecan 3.x.x) also contain MATLAB code snippet for each function – instead of searching through the MATLAB driver editor simply copy/paste from help file:


R&S Spectrum Analyzers VXI Plug&Play Instrument Driver Help
Hide
Back
Print
Options
Contents
Index
Search
Welcome
Why Instrument Drivers?
Supported Environments
Supported Interfaces
What's New?
Getting started and Examples
Instrument Driver Tree Structure
Initialize
Initialize With Options
Configuration
Set Spectrum Analysis Mode
Configure Acquisition
Channel
Common Measurement Settings
Input
Output
User Port
Frequency and Span
Configure Frequency Start Stop
Configure Frequency Center Spar
Configure Frequency Center
Configure Frequency Step Size
Configure Frequency Step Size A
Configure Horizontal Scale
Configure Frequency Span Full
Configure Frequency Span Zero
Configure Frequency Offset
Configure Frequency Coupling Fa
Configure Signal Track
Configure Frequency Mode
Amplitude
Sweep

Configure Frequency Center

C Function Prototype

```
ViStatus rsspecan_ConfigureFrequencyCenter (ViSe
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 |
|---|---|---|
| instrumentHandle | ViSession | The ViSession handle tha from the rsspecan_init or rsspecan_InitWithOptions handle identifies a particu session.<br>Default Value: None |
| reserved | ViInt32 | This control is reserved. |
| centerFrequency | ViReal64 | The center frequency of t sweep.<br>Units: Hertz<br>Default Value: None<br>Attribute:<br>RSSPECAN_ATTR_FREQU |

*Figure 26: MATLAB Function code snippets in the instrument driver help file*

If the version of VXI plug&play help file doesn't contain the MATLAB prototypes, they can be found in MATLAB Instrument driver editor (see Figure 18).

## 3.2 Setting a Property value

Setting any property value always looks like this:

```
invoke(deviceObj, 'SetProperty', PropertyID, ValueToSet,
nm_RepCap);
```

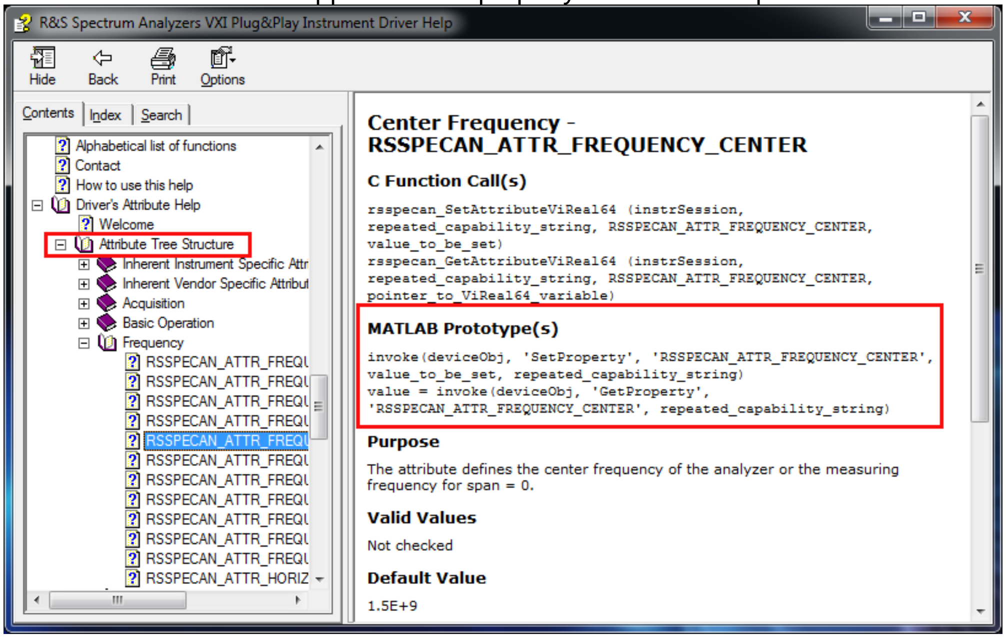
Example for setting the property RSSPECAN_ATTR_FREQUENCY_CENTER:

```
invoke(deviceObj, 'SetProperty', 'ATTR_FREQUENCY_CENTER',
3.25E+9, 'Win1');
```

As you can see it's just another function called `'SetProperty'`.
It has 3 parameters (2 mandatory):

- `PropertyID` : See 3.4 for different possibilities on how to address properties.
- `ValueToSet` : value to be set. Its type differs with different attribute types (double, integer, string, boolean)
- `nm_RepCap` : This is non-mandatory variable (all non-mandatory variables have prefix `nm_`). If omitted or equal to `'...'`, it is taken from previous setting of `set(deviceObj, 'RepCapIdentifier', repcapstring);` See Figure 28 for where to find supported RepCaps for certain property.

New versions of R&S instrument drivers (rsspecan 3.x.x Attribute Tree structure) also contain MATLAB  code snippet for each property – Set and Get parts:



**Center Frequency - RSSPECAN_ATTR_FREQUENCY_CENTER**

**C Function Call(s)**

```
rsspecan_SetAttributeViReal64 (instrSession,
repeated_capability_string, RSSPECAN_ATTR_FREQUENCY_CENTER,
value_to_be_set)
rsspecan_GetAttributeViReal64 (instrSession,
repeated_capability_string, RSSPECAN_ATTR_FREQUENCY_CENTER,
pointer_to_ViReal64_variable)
```

**MATLAB Prototype(s)**

```
invoke(deviceObj, 'SetProperty', 'RSSPECAN_ATTR_FREQUENCY_CENTER',
value_to_be_set, repeated_capability_string)
value = invoke(deviceObj, 'GetProperty',
'RSSPECAN_ATTR_FREQUENCY_CENTER', repeated_capability_string)
```

**Purpose**

The attribute defines the center frequency of the analyzer or the measuring frequency for span = 0.

**Valid Values**

Not checked

**Default Value**

1.5E+9

*Figure 27: MATLAB Property code snippets in the instrument driver help file*

## 3.3 Getting a Property value

Getting any property value always looks like this:

```
ValueRead = invoke(deviceObj, 'GetProperty', PropertyID,
nm_RepCap, nm_BufferSize);
```

Example for setting the property RSSPECAN_ATTR_FREQUENCY_CENTER:

```
SPAfrequencyCenter = invoke(deviceObj, 'GetProperty',
'ATTR_FREQUENCY_CENTER', 'Win1');
```

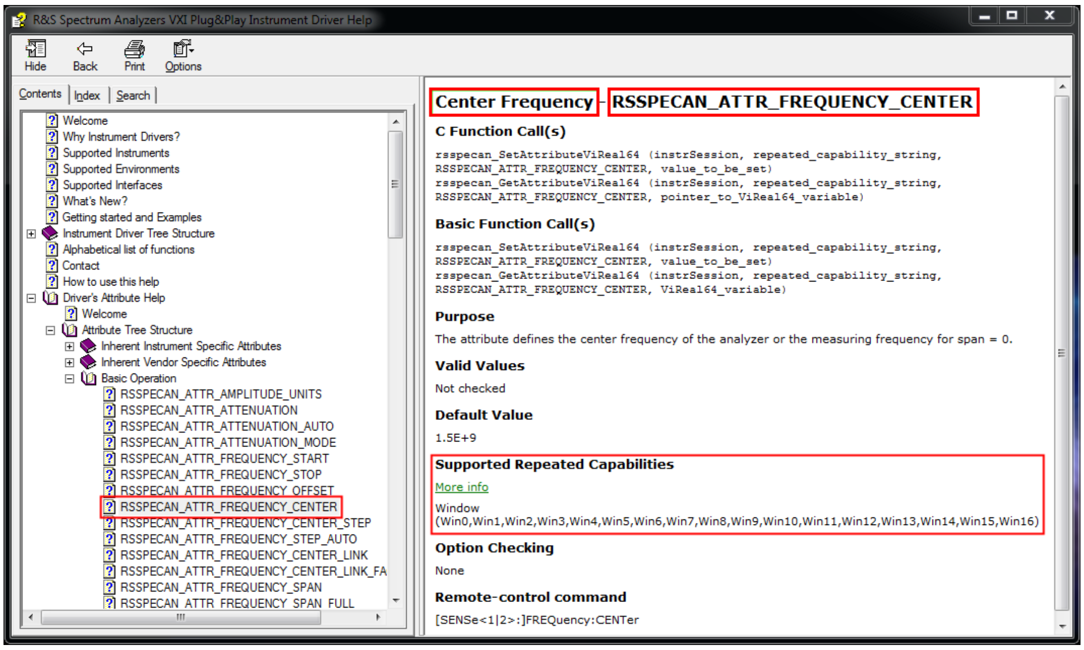
As you can see it's just another function called `'GetProperty'`.
It has 3 parameters (1 mandatory) and one return value:

- `PropertyID` : See 3.4 for different possibilities on how to address properties.
- `nm_RepCap` : Same as for `'SetProperty'`, non-mandatory
- `nm_BufferSize` : This is non-mandatory variable, which is only utilised in case of string properties. It defines the expected size of a string value. If omitted, it is set to 4096 bytes.
- `ValueRead` : Return value of the property

See the Figure 27 for the instrument driver help MATLAB code snippet.

## 3.4 PropertyID - Property Identificator

Property Identificator offers several options on how to address the Property. Let's have a look at the VXI plug&play instrument driver help (see 2.4.1 for more details) for property (attribute) RSSPECAN_ATTR_FREQUENCY_CENTER (Figure 28):



- Welcome
- Why Instrument Drivers?
- Supported Instruments
- Supported Environments
- Supported Interfaces
- What's New?
- Getting started and Examples
- Instrument Driver Tree Structure
- Alphabetical list of functions
- Contact
- How to use this help
- Driver's Attribute Help
  - Welcome
  - Attribute Tree Structure
    - Inherent Instrument Specific Attributes
    - Inherent Vendor Specific Attributes
    - Basic Operation
      - RSSPECAN_ATTR_AMPLITUDE_UNITS
      - RSSPECAN_ATTR_ATTENUATION
      - RSSPECAN_ATTR_ATTENUATION_AUTO
      - RSSPECAN_ATTR_ATTENUATION_MODE
      - RSSPECAN_ATTR_FREQUENCY_START
      - RSSPECAN_ATTR_FREQUENCY_STOP
      - RSSPECAN_ATTR_FREQUENCY_OFFSET
      - RSSPECAN_ATTR_FREQUENCY_CENTER
      - RSSPECAN_ATTR_FREQUENCY_CENTER_STEP
      - RSSPECAN_ATTR_FREQUENCY_STEP_AUTO
      - RSSPECAN_ATTR_FREQUENCY_CENTER_LINK
      - RSSPECAN_ATTR_FREQUENCY_CENTER_LINK_FA
      - RSSPECAN_ATTR_FREQUENCY_SPAN
      - RSSPECAN_ATTR_FREQUENCY_SPAN_FULL

**Center Frequency - RSSPECAN_ATTR_FREQUENCY_CENTER**

**C Function Call(s)**

```
rsspecan_SetAttributeViReal64 (instrSession, repeated_capability_string,
RSSPECAN_ATTR_FREQUENCY_CENTER, value_to_be_set)
rsspecan_GetAttributeViReal64 (instrSession, repeated_capability_string,
RSSPECAN_ATTR_FREQUENCY_CENTER, pointer_to_ViReal64_variable)
```

**Basic Function Call(s)**

```
rsspecan_SetAttributeViReal64 (instrSession, repeated_capability_string,
RSSPECAN_ATTR_FREQUENCY_CENTER, value_to_be_set)
rsspecan_GetAttributeViReal64 (instrSession, repeated_capability_string,
RSSPECAN_ATTR_FREQUENCY_CENTER, ViReal64_variable)
```

**Purpose**

The attribute defines the center frequency of the analyzer or the measuring frequency for span = 0.

**Valid Values**

Not checked

**Default Value**

1.5E+9

**Supported Repeated Capabilities**

More info

Window
(Win0,Win1,Win2,Win3,Win4,Win5,Win6,Win7,Win8,Win9,Win10,Win11,Win12,Win13,Win14,Win15,Win16)

**Option Checking**

None

**Remote-control command**

[SENSe<1|2>:]FREQuency:CENTer

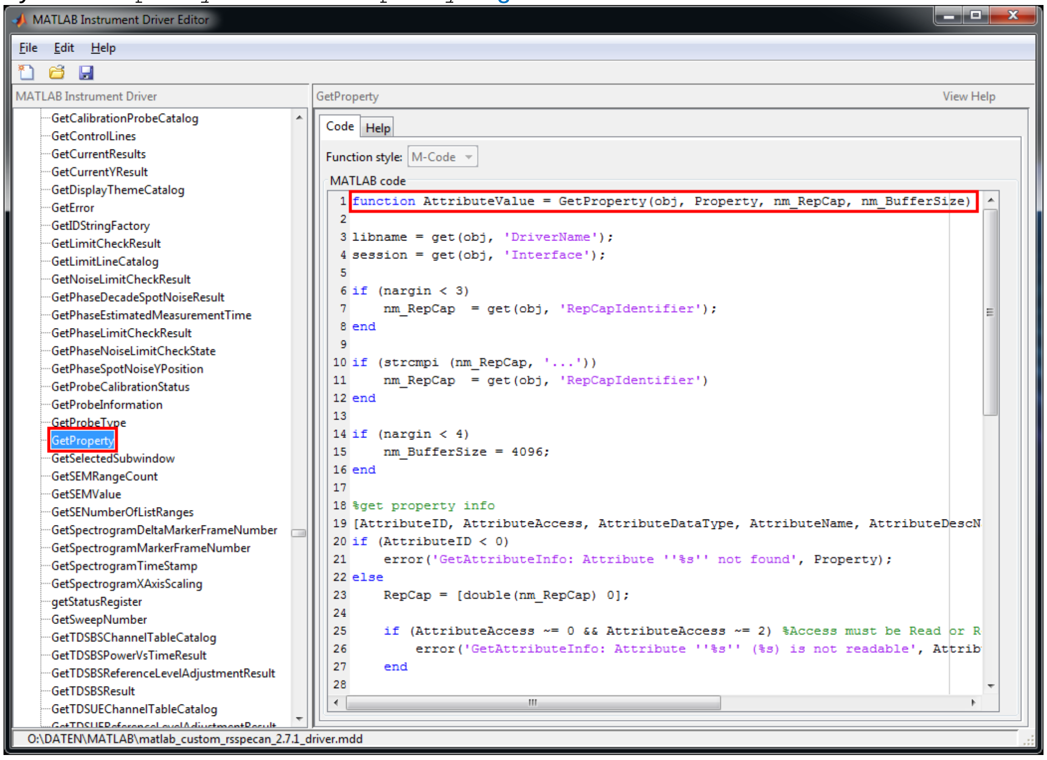
***Figure 28: rsspecan instrument driver help property RSSPECAN_ATTR_FREQUENCY_CENTER – highlighted are property identification strings and supported repeated capabilities***

In case of RSSPECAN_ATTR_FREQUENCY_CENTER, property identificator `PropertyID` can be:

- Case Insensitive String value `'RSSPECAN_ATTR_FREQUENCY_CENTER'` - complete name including the prefix
- Case Insensitive String value `'ATTR_FREQUENCY_CENTER'` - name without the prefix
- Case Insensitive String value `'Center Frequency'` – descriptive name
- Case Insensitive String value `'Center_Frequency'` – descriptive name with underscores instead of spaces
- Integer number found in rsspecan.h for RSSPECAN_ATTR_FREQUENCY_CENTER: `1150009`

## 3.5 For advanced users

MATLAB instrument driver file (mdd file) is an XML file with defined structure that MATLAB parses during the `icdevice` object creation. Using MATLAB instrument driver editor (see 2.3) you can modify the functionality for setting (function `SetProperty`) or getting (function `GetProperty`) the attribute values. Also, you can see the function `GetAttributeInfo` code for obtaining attribute info which is used by `SetProperty` and `GetProperty`: Figure 29



Function style: M-Code

```
function AttributeValue = GetProperty(obj, Property, nm_RepCap, nm_BufferSize)

libname = get(obj, 'DriverName');
session = get(obj, 'Interface');

if (nargin < 3)
    nm_RepCap  = get(obj, 'RepCapIdentifier');
end

if (strcmpi (nm_RepCap, '...'))
    nm_RepCap  = get(obj, 'RepCapIdentifier')
end

if (nargin < 4)
    nm_BufferSize = 4096;
end

%get property info
[AttributeID, AttributeAccess, AttributeDataType, AttributeName, AttributeDescN
if (AttributeID < 0)
    error('GetAttributeInfo: Attribute ''%s'' not found', Property);
else
    RepCap = [double(nm_RepCap) 0];

    if (AttributeAccess ~= 0 && AttributeAccess ~= 2) %Access must be Read or R
        error('GetAttributeInfo: Attribute ''%s'' (%s) is not readable', Attrib
    end

```

O:\DATEN\MATLAB\matlab_custom_rsspecan_2.7.1_driver.mdd

*Figure 29: GetProperty function of custom rsspecan mdd MATLAB driver*

# 4 Using MATLAB Interface for External Libraries

Although using MATLAB Instrument drivers offers several advantages, especially convenient operation with strings and error handling by generating exceptions, it is possible to use Rohde & Schwarz VXIplug&play instrument drivers with basic MATLAB package. It supports an interface for using 32-bit and 64-bit external libraries. This functionality is called ***calllib*** interface.

Rohde & Schwarz VXIplug&play instrument drivers come with a dynamic linked library (DLL). As a result, the remote-control drivers can be used directly without utilizing the Instrument Control Toolbox. This approach is described in the following chapters.

## 4.1 calllib: An interface to Generic Libraries

The ***calllib*** interface makes it easy to access already existing dynamic linked libraries. MATLAB supports this feature for 32-bit and 64-bit libraries on Windows operating systems.

Please note:

- in 32-bit operating system you can only use 32-bit version of MATLAB and 32-bit type of VXIplug&play instrument driver
- in 64-bit operating system you can use 32-bit or 64-bit version of MATLAB.

The version of VXIplug&play driver always has to match the version of MATLAB, not the version of operating system. That means, for 32-bit MATLAB only the 32-bit version of VXIplug&play driver can be used and for 64-bit MATLAB only the 64-bit version of VXIplug&play driver can be used. To avoid confusion it is recommended to install only one version of VXIplug&play driver.

## 4.2 Installing VXIplug&play Instrument Drivers

Having an installed VXIplug&play instrument driver on the host PC is a prerequisite before remote controlling any instrument from MATLAB by using instrument driver.

Rohde & Schwarz VXIplug&play instrument drivers and installation manuals are available in the Drivers download area on the Rohde & Schwarz website: http://www.rohde-schwarz.com/drivers/.

## 4.3 Loading and Acquiring Information about Libraries

Generally any library needs to be loaded into MATLAB before it is used. It is important to propagate the path where the libraries are accessible to MATLAB (see example scripts rsspecan_MatlabCalllib_32bit.m or rsspecan_MatlabCalllib_64bit.m). They show how to load a VXIplug&play rsspecan instrument driver library. In this example, the rsspecan instrument driver is used. The variables *vxipnpLib* and *vxipnpLibDll* can be reassigned to refer any other Rohde & Schwarz VXIplug&play instrument driver library. After this step, the referred library is ready to be used in MATLAB .

### 4.3.1 Acquiring information about the libraries

The following functions are available for acquiring more information about the library and its application programming interface (API):

```
>> libfunctions (vxipnpLib, '-full');
>> libfunctionsview (vxipnpLib);
```

The most important information is that about the signatures of the library function calls. This information makes it easy to gain an understanding of necessary function parameter types.

## 4.4 Calling Library Functions

Calling functions from libraries is a critical task, because MATLAB has a different data representation than the ANSI C programming language. As an example, let's take a function with which a trace from a Rohde & Schwarz spectrum analyzer can be captured: `rsspecan_ReadYTrace`

C-prototype of the function can be found in instrument driver help file:

```
ViStatus err = rsspecan_ReadYTrace (ViSession instrumentHandle,
ViInt32   window, ViInt32 trace, ViUInt32 maximumTime_ms,
ViInt32 arrayLength, ViInt32* amplitude, ViReal64[] SPAcoorsY);
```

MATLAB code snippet with the same functionality:

```
SPAwindow                = 0;
SPAtrace                 = 1;
SPAtimeoutMs             = 5000;
SPAarrayLen              = SPAsweepPoints;
SPAarrayActualLen        = -1;
SPAcoorsY                = zeros(SPAarrayLen, 1);

[err, SPAarrayActualLen, SPAcoorsY] = calllib(vxipnpLib,
'rsspecan_ReadYTrace', SPAsession, SPAwindow, SPAtrace,
SPAtimeoutMs, SPAarrayLen, SPAarrayActualLen, SPAcoorsY);
```

Background colors are matching the same parameters. All functions that in C pass value through the pointers (in our case `SPAarrayActualLen` and `SPAcoorsY`) must be mentioned in output parameters as well if we want to read their values in the same order that are declared. First parameter is always return value of the function, in our case `err`. If the return value is not needed, it cannot be omitted, but it must be replaced by '~'.

If we want to retrieve only `SPAcoorsY` but not `SPAarrayActualLen` then the MATLAB code would be:

```
[err, ~, SPAcoorsY] = …
```

If `SPAcoorsY` would not be required, the MATLAB code would look like this:

```
[err, SPAarrayActualLen]= …
```

Another specialty of MATLAB is passing strings. It must be done by using ***libpointer*** function. Also, this value is always returned as other values passed by pointer. Example can be `rsspecan_init`.

C-prototype:

```
ViStatus err = rsspecan_init (ViRsrc resourceName,
ViBoolean idQuery, ViBoolean resetDevice,
ViSession* instrumentHandle);
```

`resourceName` (it’s MATLAB equivalent `pSPAresource`) is a value passed by pointer, therefore it must be counted with in return parameters, because we want to retrieve `SPAsession` value which is in order of parameters at the very end. Since `pSPAresource` is not required to be read back (its value is not changed by this function), its place can be replaced with `~`:

MATLAB code snippet:

```
SPAsession   = -1;
SPAidQuery   = 1;
SPAreset     = 1;
SPAresource = 'TCPIP::10.85.0.68::INSTR';
pSPAresource = libpointer( 'int8Ptr', [int8( SPAresource ) 0] );

[err, ~, SPAsession] = calllib(vxipnpLib, 'rsspecan_init',
pSPAresource, SPAidQuery, SPAreset, SPAsession);
```

In case of required return value, the MATLAB script is the following:

```
[err, SPAresource, SPAsession] = calllib(vxipnpLib,
'rsspecan_init', pSPAresource, SPAidQuery, SPAreset,
SPAsession);
```

Please note, that the return value variable is `SPAresource`, not `pSPAresource`.

Good example of handling string as inputs and outputs are functions `rsspecan_WriteInstrData`, `rsspecan_ReadInstrData`, `rsspecan_QueryViString` used in rsspecan_MatlabCalllib_32bit.m or rsspecan_MatlabCalllib_64bit.m.

For more detailed examples of library calling conventions, refer to the MATLAB ***calllib*** documentation.

Cleaning before exiting is done using the following MATLAB code:

```
%% clean up before exit
unloadlibrary(vxipnpLib);
clear all;
```

## 4.5 Complete application example using calllib

This example is available in attached files as rsspecan_MatlabCalllib_32bit.m or rsspecan_MatlabCalllib_64bit.m.

## 4.6 Known Problems

**Failed to preprocess the input file….**

If this error message is shown in 32-bit version of MATLAB , make sure you have MATLAB Lcc-win32 compiler selected. Run:

```
>> mex -setup
```

and follow the instructions, usually it is the choice nr. 1. Restart MATLAB

**"A 'Selected' compiler was not found. You may need to run >> *mex -setup*"**

Run:

```
>> mex -setup
```

and follow the instructions.

To successfully load a library using the ***loadlibrary*** functionality the default compiler needs to be set up in MATLAB . The supported compilers of the currently available releases are listed here:
***http://www.mathworks.de/support/compilers/current_release/win32.html***
and here:
***http://www.mathworks.de/support/compilers/current_release/win64.html***.

In this application note the **Microsoft Visual C++ 2008 SP1 V9.0 Professional Edition** compiler was used. Especially on MATLAB 64-bit installations a non-MATLAB compiler needs to be configured manually. After condiguration, restart MATLAB

# 5 References

- MathWorks Documentation: **Instrument Control Toolbox**
  ***http://www.mathworks.com/access/helpdesk/help/toolbox/instrument/***
  Retrieved: February 2014

- MathWorks Documentation: Shared Libraries (***calllib***)
  ***http://www.mathworks.com/access/helpdesk/help/techdoc/ref/calllib.html***
  Retrieved: February 2014

- National Instruments **VISA 5.4** download page for Windows:
  ***http://www.ni.com/download/ni-visa-run-time-engine-5.4/4231/en/***
  You need to be registered to be able to download the package.
  Retrieved: February 2014

# 6 Attached Files

Please refer to ***1MA171_files.zip*** file that is attached to this application note. It contains the following files:

## 6.1 raw_SCPI_communication.m

Simple example of MATLAB script reading ***IDN?** response from the instrument using MATLAB VISA Interface Object.

## 6.2 rsspecan_TMTOOL_functions_example.m

This MATLAB script is a result of TMTOOL ***Session Log*** when using the function for settings the spectrum analyser centre frequency.

## 6.3 rsspecan_TMTOOL_properties_example.m

This MATLAB script is partially a result of TMTOOL ***Session Log*** when using the function for settings the spectrum analyser centre frequency.

Under comments:
```
% Alternative way to set the property
```
it is shown in addition, how the same results can be achieved by using the function
```
setattributevireal64
```

## 6.4 rsspecan_TMTOOL_direct_functions_example.m

This example shows the result MATLAB script composed in chapter 2.7.1.

Description:
```
%% MATLAB example for direct composing a TMTOOL script using
functions. The example configures spectrum analyzer center
frequency, sets single sweep, queries *IDN? string, performs the
sweep and reads the trace data.
```

## 6.5 rsspecan_TMTOOL_complex_example.m

This example shows how to use TMTOOL MATLAB instrument driver to perform settings, measurement synchronizations, reading the results and handling errors. Most of other applications and tasks for spectrum analyzer, but also for other instruments can be derived from this example. In several cases it shows the same functionality with different approaches for the user to choose his preferred one.

It is also from the functional point of view equivalent to rsspecan_calllib_complex_example_32bit.m, so the user can compare them.

Description:
```
MATLAB TMTOOL example for R&S Spectrum Analyzer Instrument
Driver rsspecan. The example configures spectrum analyzer,
performs the sweep, reads the trace data from one sweep and then
from 5 maxhold / minhold faster sweeps from 2 different traces.
Then, makes a screenshot of the analyzer screen, copies it to
the PC and displays it.
```

## 6.6 rsspecan_calllib_complex_example_32bit.m

This example shows how the MATLAB calllib can be utilized to perform the same tasks as the script rsspecan_TMTOOL_complex_example.m. Notice the different approach to error handling and operations with strings. Use only with the 32-bit MATLAB versions.

Description: Same as in 6.5

## 6.7 rsspecan_calllib_complex_example_64bit.m

Same MATLAB script as in 6.6 for 64-bit MATLAB versions. The only difference is at the beginning when loading the library. Use only with the 64-bit MATLAB versions.

## 6.8 rsspecan_MatlabCalllib_32bit.m

MATLAB script for loading the 32-bit ***rsspecan_32.dll*** library and showing functions prototypes. Use only with the 32-bit MATLAB versions.

## 6.9 rsspecan_MatlabCalllib_64bit.m

MATLAB script for loading the 64-bit ***rsspecan_64.dll*** library and showing functions prototypes. Use only with the 64-bit MATLAB versions.
Note the reference to workaround_64bit_rsspecan.h instead of ***rsspecan.h***. This is due to not supported calling conventions in MATLAB.

## 6.10 workaround_64bit_rsspecan.h

This is the workaround file for loading the rsspecan library in 64-bit MATLAB with the following content:

```
#undef _LDSUPPORT
#define __fastcall
#include "rsspecan.h"
```

It overcomes the limitation of not supported calling conventions in MATLAB 64-bit. The file should be copied to the same directory as ***rsspecan.h***: ***c:\Program Files\IVI Foundation\VISA\Win64\Include***

## 6.11 rsnrpz_TMTOOL_example.m

MATLAB TMTOOL example of communication with NRP-Z Powersensor – one shot measurement and scope mode measurement according the ***1GP69***.

Description:

```
% MATLAB TMTOOL example for rsnrpz Powersensors driver
% Follow the http://www.rohde-schwarz.com/appnote/1GP69 -
Powersensor Programming guide application note
```

## 6.12 rsnrpz_calllib_example_32bit.m

This example shows how the MATLAB calllib can be utilized to perform the same tasks as the previously described script rsnrpz_TMTOOL_example.m. Use only with the 32-bit MATLAB versions.

Description: Same as in 6.11

## 6.13 rsnrpz_calllib_example_64bit.m

Same MATLAB script as in 6.12 for 64-bit MATLAB versions. The only difference is at the beginning when loading the library. Use only with the 64-bit MATLAB versions.

## 6.14 CUSTOM_rsspecan_TMTOOL_Complex_example.m

Functionally the same script as rsspecan_TMTOOL_complex_example.m but with the use of custom rsspecan mdd driver. Notice the device object `deviceObj` used in all functions and properties.
Notice that the block `%% Commonly used group objects` is completely missing

Description:

```
% Differences to MATLAB makemid driver:
% - No group object, only device object is used for accessing
all functions
% - Function names are exactly the same as in rsspecan_vxi.chm
driver help
% (simply copy and paste), but the prefix rsspecan_ is removed
% - all new R&S VXIpnp drivers: help files will contain snippets
of
% MATLAB code for functions and properties
% - Accessing properties is easier with more possibilities to
find and address them
% - rsspecan_vxi.chm help file is intended to be used, so mdd
file contains no help fields
% - R&S custom rsspecan mdd driver is cca 8x smaller than output
file from makemid
```

## 6.15 CUSTOM_rsnrpz_TMTOOL_example.m

Functionally the same script as rsnrpz_TMTOOL_example.m but with the use of custom rsnrpz mdd driver. No setting or reading the property values is possible with this driver, since rsnrpz is not attribute-based driver.

Notice that the block `%% Commonly used group objects` is completely missing

# 7 Additional Information

Please send your comments and suggestions regarding this application note to:

***TM-Applications@rohde-schwarz.com***

Using tag **"[1MA171]"** in the mail subject will help us to quickly identify the topic and speed up the response process.