How to use Rohde & Schwarz® Instruments with LabVIEW™ Application Note

Having the possibility to remotely control instruments has become the necessity not only in the field of automated production testing but more and more already in the phase of development. One of the programming environments that makes this task accessible for designers with very little or no programming experience is LabVIEW. To make the remote control of instruments easier, Rohde & Schwarz provides the LabVIEW Instrument drivers that take away the burden of controlling, synchronization, response formatting and error handling from the developer.

This paper explains how to use Rohde & Schwarz instruments together with LabVIEW in order to prepare an automated measurement task quickly and efficiently. It focuses mainly on using Rohde & Schwarz LabVIEW Instrument drivers, in addition in several occasions it also shows comparisons with the approach of raw SCPI communication.

Note:

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

www.rohde-schwarz.com/appnote/1MA228

Instrument Drivers LabVIEW – 1MA228_1e

Miloslav Macko

Application Note

ROHDE&SCHWARZ

Contents

1 Introduction

Rohde & Schwarz provides a range of free Instrument Drivers (further referred to as '**drivers**') (follow the link Rohde & Schwarz drivers) to simplify the development process of instrument remote control applications. Among them are LabVIEW attribute-based drivers. These drivers utilize the Express VI technology which internally takes advantage of VI scripting - programmatically creating and modifying VIs. In all new Rohde & Schwarz drivers releases we introduce the **Fast Read** and the **Fast Write** possibility. This feature removes one common disadvantage that has discouraged users from using drivers - speed of execution. More about this topic is discussed in chapter 4, "Driver Express VI", on page 14 and chapter 5, "Performance comparison", on page 32.

For demonstration purposes the Rohde & Schwarz LabVIEW driver for Spectrum Analyzers (`rsspecan`) is used in this Application Note. However, the presented procedure is applicable to all Rohde & Schwarz drivers. The Spectrum Analyzer driver is chosen because it represents the most comprehensive application when communicating with any other instrument – settings, waiting for the measurement result, reading the results either in strings or arrays of numbers.

1.1 Required Software

To follow the steps described in this Application Note the following software is required:

- Windows XP/Vista/7 32-bit/64-bit operating system
- LabVIEW 2010 or later 32-bit/64-bit
- VISA I/O library (e.g. National Instruments VISA Version 5.x)

The configuration used in for this Application Note: Windows 7 64-bit, LabVIEW 2010 64-bit, NI VISA 5.4.0, used driver in all examples is rsspecan LabVIEW driver version 3.1.0, 07/2014, Attribute Express VI version 1.10

1.2 Related Documents

The Application Notes discussing remote-control drivers and their usage:

- 1MA153: Development Hints and Best Practices for Using Instrument Drivers
- 1MA170: Introduction to Attribute Based Instrument Drivers

2 About LabVIEW drivers

2.1 LabVIEW driver types

Over the time Rohde & Schwarz LabVIEW driver technologies have gone through several phases of improvement due to internal and external factors (more instrument capabilities and higher complexity, driver specification changes...). Currently, Rohde & Schwarz LabVIEW drivers can be split into three groups (in order from older to newer):

2.1.1 Non-attribute based driver

This type has direct `VISA Write.vi` and `VISA Read.vi` inside the function VI. This approach is not used anymore, because they are difficult to maintain, they don't perform error checking and if used with modern spectrum analyzers or signal generators, the driver would contain thousands of VIs. (e.g. rsrfsiggen driver has over 4000 attributes which would result in over 8000 VIs (read/write).

Installation: Copy the `instr.lib` folder to
`c:\Program Files\National Instruments\LabVIEW 2010\`

The folder `user.lib` is either not present in the installation package or it is empty.

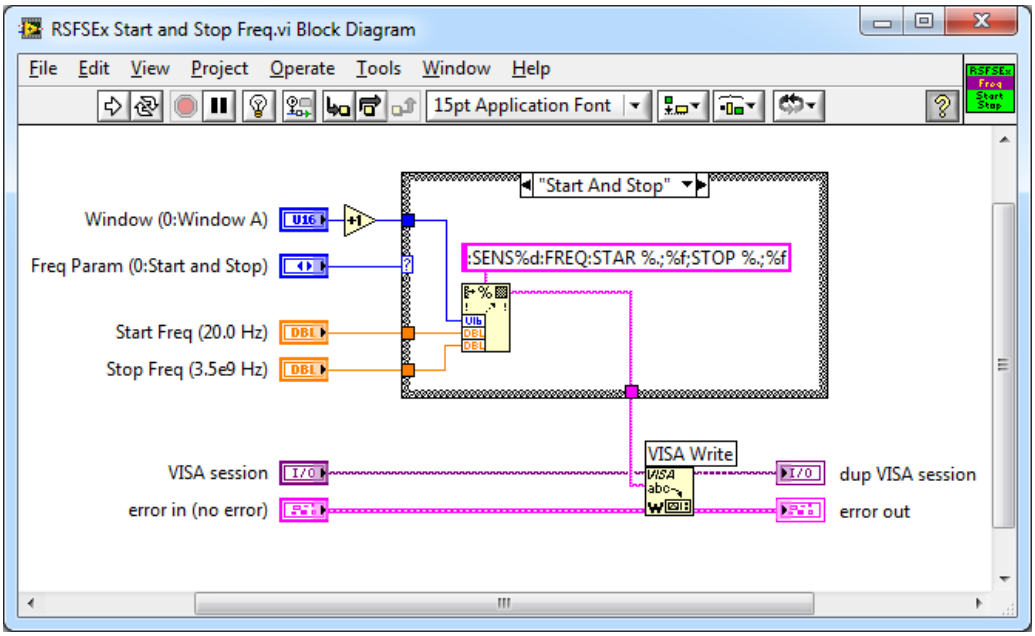


Fig. 2-1: Example of non-attribute based FSEx driver function VI.

2.1.2 DLL-wrapper around the VXI plug&play driver

This type of driver uses the VXI plug&play `dll` library. Although there are some advantages to this approach (common source code for all types of drivers, faster execution speed, smaller load on memory and disk space), the main disadvantage is that it's not a LabVIEW code and the user cannot see it or alter it. Therefore this type is now only used for older instruments and for the ones that use proprietary communication protocols without VISA support (e.g. USB NRP-Z power sensors).The dll-wrapper drivers are easily recognizable, by `msi` installation package, since they always require installation of VXI plug&play driver.

Installation: The driver installation is a `msi` package that installs VXI plug&play driver and LabVIEW VIs to:
`c:\Program Files\IVI Foundation\VISA\GWin64\<driver_name>`

To have the access to the driver palette, copy the folder `<driver_name>` to `c:\Program Files\National Instruments\LabVIEW 2010\instr.lib\` folder and restart LabVIEW.

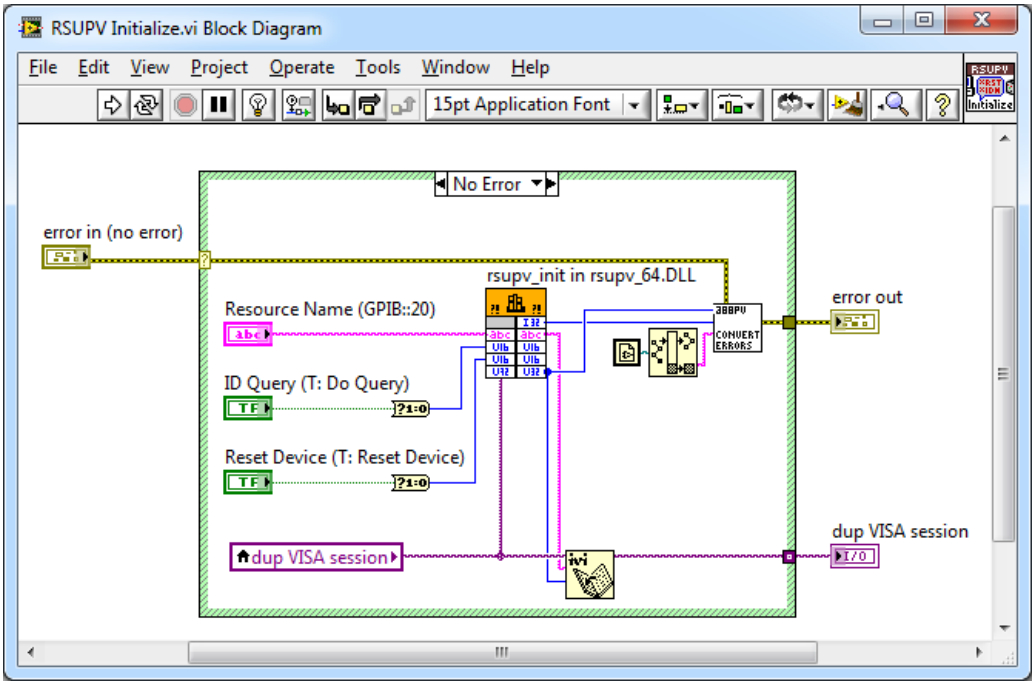


Fig. 2-2: Example of dll-based RSUPV driver.

2.1.3 Attribute-based driver

This type is the newest one that is also the subject of this Application Note. Instrument functionalities are split into attributes which can be set or read separately using Attribute Express VIs (further referred to as '**Express VI**'). Configured Express VIs (Express VI instances) are used in the driver **Hi-level functions** (see the example of such Hi-level function below). In occasions, custom LabVIEW code is used when the desired

functionality cannot be performed by attributes (for example reading the analyzer trace, setting/reading a property that contains a cluster of parameters).

Installation: Copy both folders `instr.lib` and `user.lib` to `c:\Program Files\National Instruments\LabVIEW 2010\`

Folder `instr.lib` contains the driver Hi-level functions, `user.lib` contains the driver's Express VI.

Example of an attribute-based driver Hi-level function for setting center frequency and span using two Express VI instances (enclosed by red rectangle):

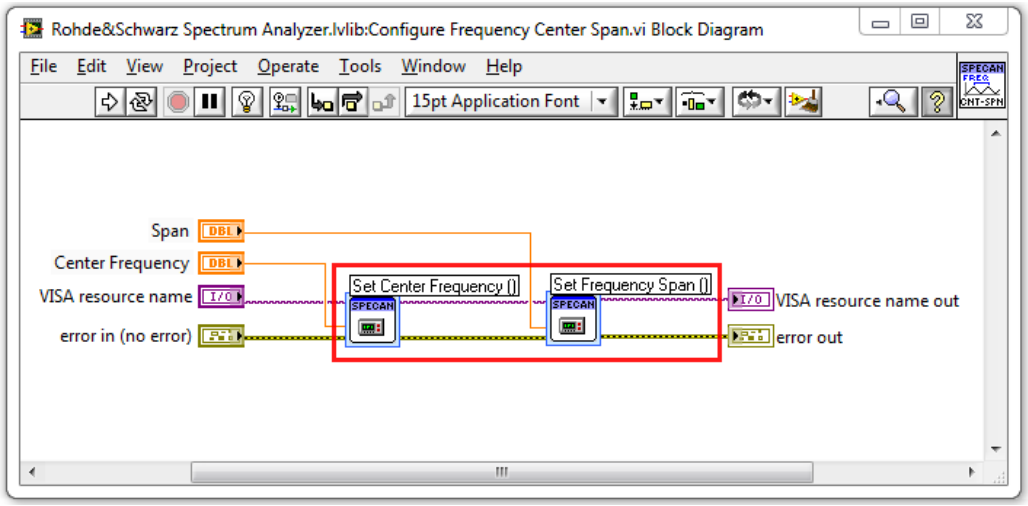


Fig. 2-3: Example of attribute-based rsspecan driver Hi-level function VI for setting center frequency and span using two Express VI instances.

After restarting LabVIEW, the driver functions are accessible through palette menu:

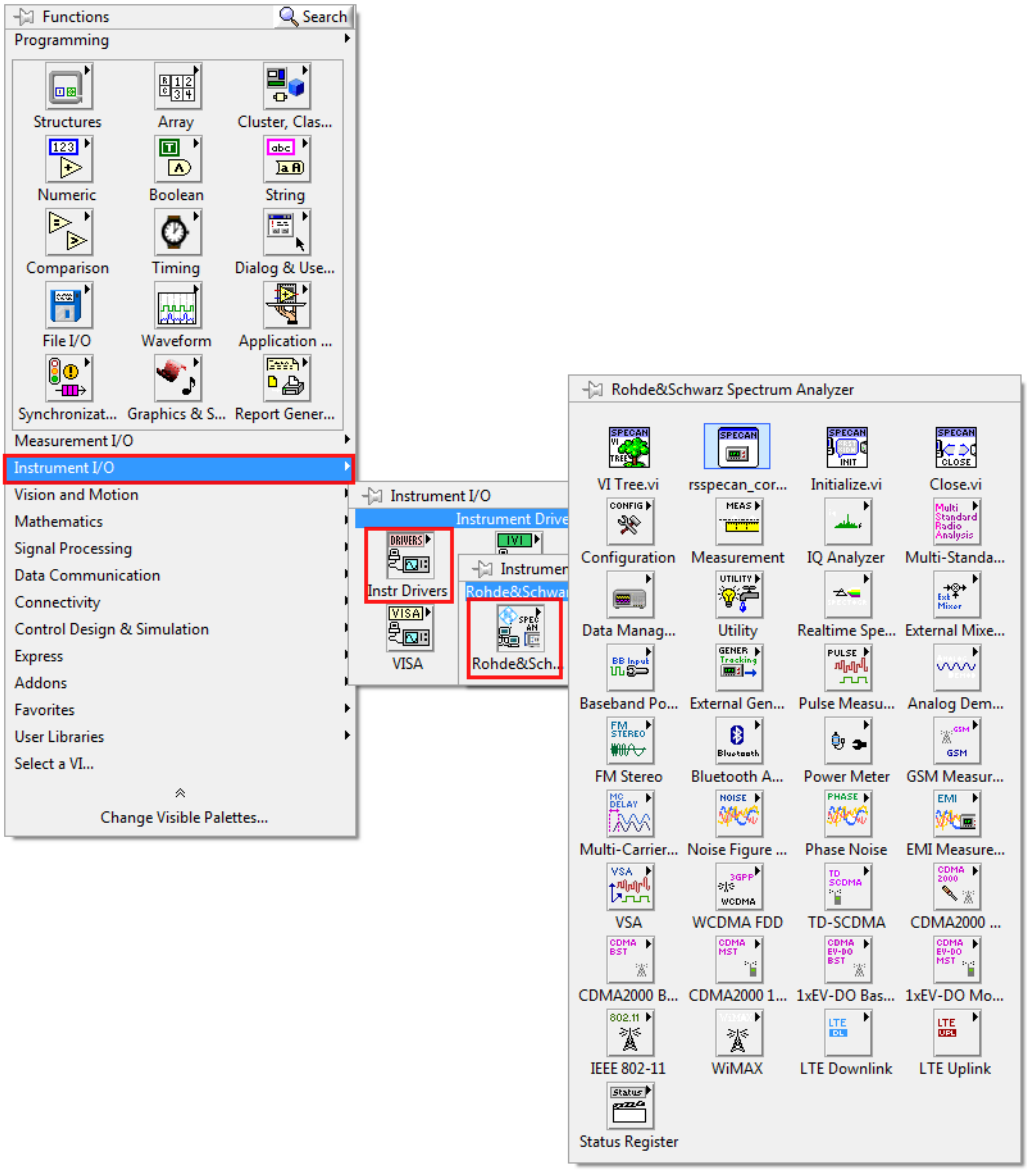


Fig. 2-4: Rohde & Schwarz Spectrum Analyzer driver in LabVIEW palette view under Instrument I/O -> Instrument Drivers -> Rohde & Schwarz Spectrum Analyzer.

Palette menu node pictures (e.g. **Configuration**) are stored in `dir.mnu` files in every driver folder and sub-folders. The tree-structure in the palette corresponds to the help file (Microsoft compressed html file format *.chm) provided for every Rohde & Schwarz driver:

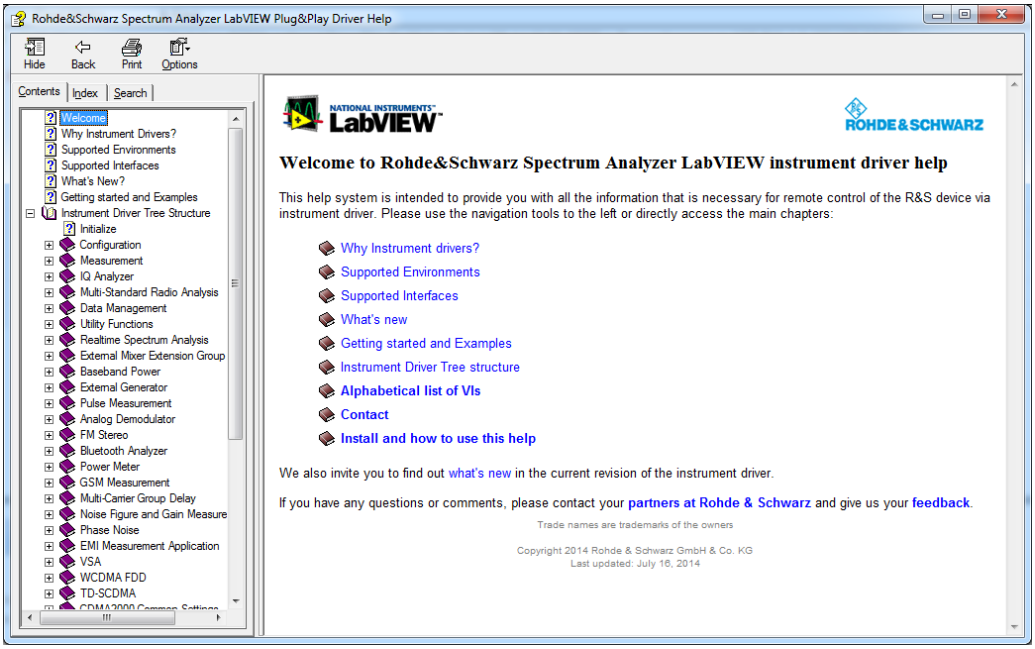


Fig. 2-5: Rohde & Schwarz Spectrum Analyzer driver help file rsspecan.chm

3 Getting started with using attribute-based drivers

3.1 Driver structure

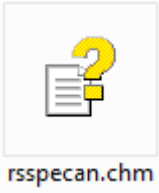
The LabVIEW driver consists of a multi-level folder and file library forming a tree-structure that corresponds to the help file Contents node **Instrument Driver Tree Structure** or shortly **VI Tree**. This structure is also reflected in `<driver_name> VI Tree.vi`.

The root folder is different for a project-based drivers (newer type) and previously used non-project-based drivers:

- **Project-based drivers** have no `<driver_name>` prefix for the VIs, because the driver prefix is the name of the project they are members of. They contain 2 folders: `Private` and `Public`. `Private` is reserved only for internal driver use, `Public` is the root folder for the user.
- **Non-Project drivers** have `<driver_name>` prefix for all VIs and there are no restrictions to the availability of VIs. `<driver_name>` folder is the root folder for the user.

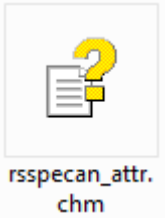
The following files can be found in the user root folder (VI names mentioned here are without `<driver_name>` prefix):

- driver help file `<driver_name>.chm`

  Microsoft CHM file that gives an overview of the driver structure: Hi-level functions, attributes (by linking the driver's attribute help file mentioned below), additional driver information. It also cross-references SCPI commands and attribute IDs, so you can find which command is used in which attribute or Hi-level function.

- driver's attribute help file `<driver_name>_attr.chm`

  Microsoft CHM file that shows the attribute tree structure which is used by the Express VI Tree control to access the desired attribute.

- `Initialize.vi`

  This is a basic VI that opens a session to the instrument based on the input VISA resource name. Output VISA session is then used throughout the driver as a handle for communication with that specific instrument. VISA session is compatible with all VISA palette VIs, including `VISA Read.vi` and `VISA Write.vi`.

  Important!: Don't use `VISA Open.vi` from LabVIEW palette to initialize a session, because `Initialize.vi` also creates additional data for the session that are needed later on.

  Optional parameters are **Reset** (defines whether to send `*RST` to the instrument after initialization) and **ID Query** (checks based on the instrument response to ***IDN?** query whether it is among the listed supported devices for this driver). In special occasions you might want to disable **ID Query** if you are sure that the driver can handle your instrument. Disable **Reset** if your instrument already has a setting that you don't want to lose.

- `Close.vi`

  This VI closes the instrument session and frees all additional session data.

  Important!: Don't use `VISA Close.vi` from LabVIEW palette to close a session, because `Close.vi` also clears all additional session data that the `Initialize.vi` has created.

- `Utility` folder (don't mix with `_utility`) is outlined in the next chapter.

3.1.1 Utility folder functions

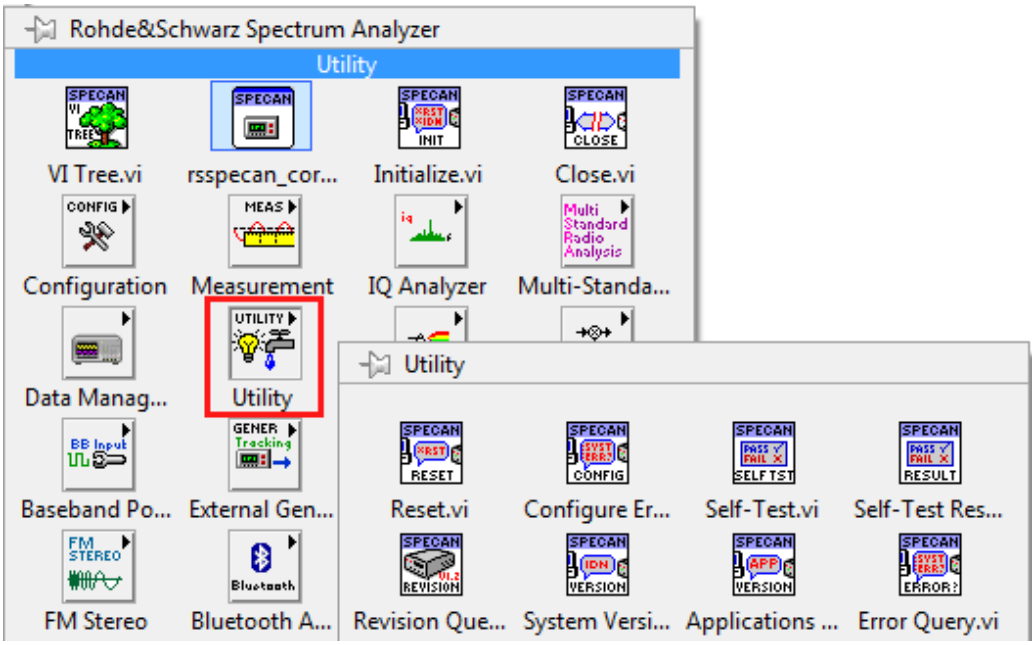


Fig. 3-1: rsspecan LabVIEW Utility palette.

This folder contains VIs that change the driver's behavior, error handling VIs, instrument reset and self-test VIs, etc... The following chapters will describe the ones most commonly used.

3.1.1.1 Instrument Status Checking.vi

This VI changes the session based parameter **ErrorChecking** which is set to **ON** during initialization. When ON, the driver calls the VI `_check_error.vi` (see below) that sends the SCPI query `*STB?` after each command/query, checks the bit 2 of the Status register (Error Queue not empty) and reports an eventual errors to the error cluster.

Set this parameter to OFF for sections of your code that require fast execution speed. Refer to chapter 5, "Performance comparison", on page 32.

3.1.1.2 Option Checking.vi

This VI changes the session based parameter **OptionChecking** which is set to **ON** during initialization. Some instrument commands require certain software or hardware options to be available. The driver checks whether a required option is available on the instrument before it sends the command.

3.1.1.3 _check_error.vi

This VI is used to detect whether an instrument has any message in its error queue. First, the VI sends SCPI query `*STB?` and checks the bit 2 (Error Queue not empty). If this bit is set to 1, `SYST:ERR?` reads and deletes the 1st entry from instrument error queue. If you used `Instrument Status Checking.vi` to set the **ErrorChecking** to OFF, there might be more than 1 errors in the queue. Therefore, in order to delete all the error messages, you must call this VI in a loop until it reports no error in error out cluster.

*Note: LabVIEW doesn't put VIs whose names start with underscore to palettes. Therefore you will not find `_check_error.vi` in Utilities LabVIEW palette. You have to use your file explorer or **Quick Drop** (CTRL+Space) to access it.*

3.1.1.4 Get Timeout.vi / Set Timeout.vi

These VIs get/set session based parameter **OPCtimeout** - waiting for the operation to be completed (e.g. measurement sweep). The driver synchronizes with the instrument by sending `*OPC` (not `*OPC?`) at the end of the command and then periodically polling the Status register bit 6 (OPC).

Important!: Because of this synchronization mechanism, the VISA Timeout parameter has no effect on the driver's measurement timeout. You need to change this custom session based parameter to prevent long measurement timeout errors.

3.1.1.5 Revision Query.vi

This VI returns the driver version and instrument firmware version.

3.1.1.6 Reset.vi

This VI sends `*RST` to the instrument, waits for the reset to be completed and applies a default instrument setup, same as during the initialization.

3.1.1.7 Instrument IO folder

This folder contains VIs for direct SCPI queries and VIs for transferring files between the PC and the instrument. Direct SCPI command writing can be performed with `VISA Write.vi` from the LabVIEW VISA palette.

4 Driver Express VI

This chapter describes the new version of the driver Express VI that will be supplied with all newly released LabVIEW drivers starting with rsspecan 3.0.0 06/2014. In case your driver has an older version that doesn't support **Fast Read/Write** operations, feel free to contact us (chapter 7, "Additional Information", on page 43) for an update.

Why should you use Express VI? There are several reasons to do so:

- **Performance**: The driver even with **ErrorChecking** switched OFF (chapter 3.1.1.1, "Instrument Status Checking.vi", on page 11) still has an overhead which in certain applications may prove too high. For these cases Express VI offers **Fast Write** and **Fast Read** operations that compose most of the SCPI command during the configuration phase. If you use fixed input parameter value, the entire SCPI command string is prepared just to perform **VISA Write** operation during the execution. Also, rather than sending SCPI commands one by one they can be composed into one string and then sent all at once to the instrument. Read more in chapter 4.2, "Express VI in drivers", on page 16 and chapter 5, "Performance comparison", on page 32.
- **Reading attribute (parameter) value**: Due to a large number of attributes, the driver doesn't provide all Hi-level functions for retrieving attribute values. For example, you can set the spectrum analyzer center frequency with `Configure Frequency Center.vi`, but there is no function to read it back. Here, you have to use Express VI configured to **Read** operation with the attribute **RSSPECAN_ATTR_FREQUENCY_CENTER**.
- **Setting just one attribute (parameter) separately**: The driver Hi-level functions are very often programmed with certain logic inside - they group setting of several attributes together. If you just wish to set one attribute, you have to use Express VI.
- **Non-availability of Hi-level function**: To minimize the size, the new Rohde & Schwarz drivers are missing Hi-level functions that set just one attribute. There are some exceptions to this for legacy reasons, or IVI specification requirements. If you cannot find a Hi-level function using the attribute you need, you can always access it by Express VI.

4.1 General principle of LabVIEW Express VI

A Express VI consists of 3 parts:

- **Execution Code VI** - VI with the actual execution code during the run-time.
- **Configuration VI** - this VI is invoked when you double-click on any Source Express VI instance in your code. The Configuration VI has a front panel and based on the settings it will modify the content of that Source Express VI instance.
- **Source Express VI** - this is the VI we refer to as **Express VI**. If you place the **Express VI** into your code, you create an**Express VI instance** (Source Express VI with a certain configuration) and this is what you see in your code. In principle, it's a wrapper over the **Execution Code VI** with configurable content (inputs, outputs,

constants, setting default values for controls...). Compared to a normal VI, the Express VI instance has 2 special properties:

– it is not saved as a separate VI, but directly inside a parent VI. Every Express VI instance is unique, therefore the more of them you have, the bigger your parent VI gets.

– it is reconfigurable by **Configuration VI**. Configuration starts when you double-click on its icon or select Context Menu → Properties. If you want to see the content of your Express VI instance, you have to convert it to a standard VI by context menu item **Open Front Panel**. However, you cannot reconfigure it again, and you have to save it separately as a standard VI. The Source Express VI is visually distinguished from standard VI by light-blue frame around it:

The following picture shows the example of a simple Express VI that can be configured to perform a math operation between 2 numbers. On a picture below, the Express VI instance is configured to give a result of A+B. Configuration VI can also take care of connecting the terminals to the Source Express VI connector and by doing so changing their accessibility to the user. To find out more on the topic of Express VI, enter "**LabVIEW Express VI Development Toolkit User Guide**" into your search engine. For the purposes of this document we will refer to the **Configuration VI** as the **Configuration panel**.

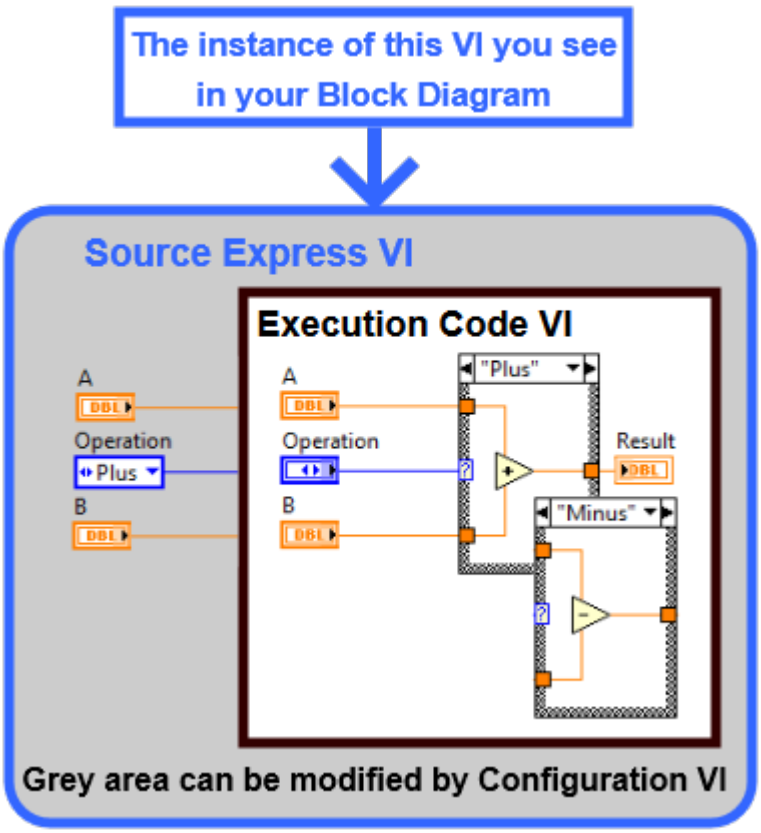


Fig. 4-1: Basic structure of Express VI. An instance of Express VI is visible in your code.

4.2 Express VI in drivers

The Express VI for the rsspecan driver is located in
`c:\Program Files\National Instruments\LabVIEW 2010\user.lib\_express\rsspecan\rsspecan_core_attribute_expressSource.llb`
library. This library contains **Source Express VI** and the **Execution Code VI** that is executed during run-time. The Express VI has always **Source** in its name. You cannot drag and drop it to the block diagram from llb (limitation of llb format). You have to use one of the following options to access the Express VI:

- **Using palettes**:

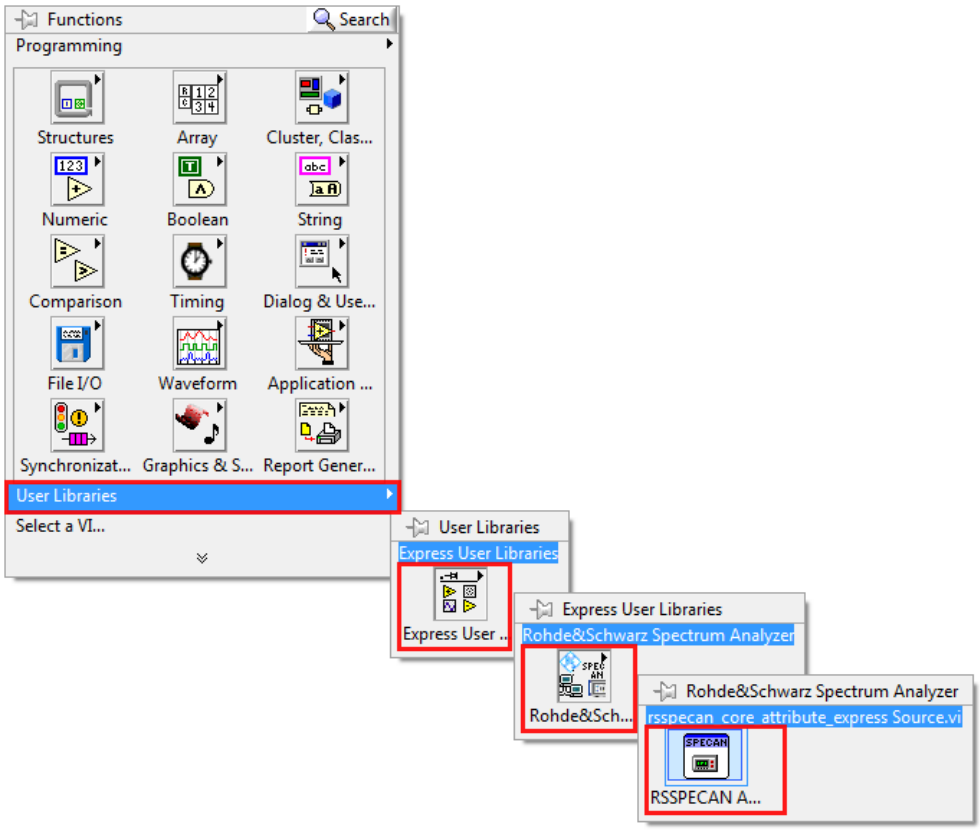


Fig. 4-2: The Express VI instance can be placed in the Block Diagram by Functions palette -> User Libraries -> Express User Libraries -> Rohde & Schwarz Spectrum Analyzer

- **Copying from existing code**: Use the driver Hi-level function e.g. figure 2-3 to copy the instances of Express VI from. Copying the Express VI instance also copies its configuration. Rich-text VI title will be replaced by plain text, because LabVIEW will add an index number at the end of the title to distinguish between the instances. You have to run the Configuration panel to get it back to rich-text title.

- **Quick Drop**: On the Block Diagram press **CTRL+Space**:

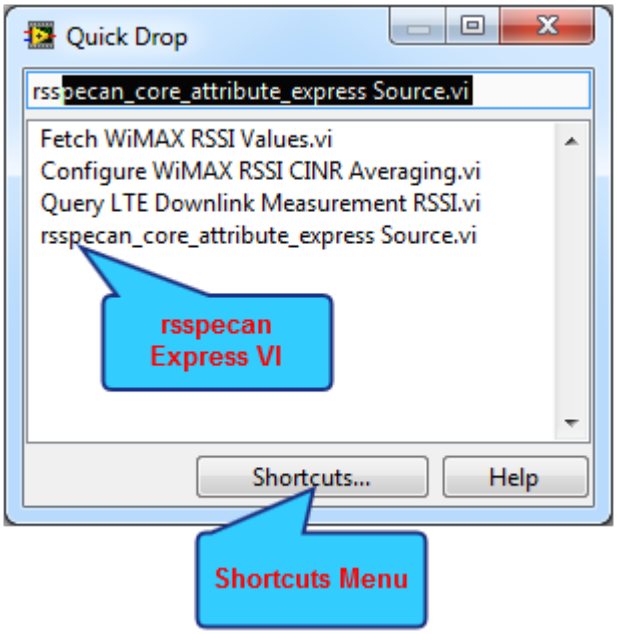


Fig. 4-3: Quick Drop window

After typing only "rss", you can already select the `rsspecan_core_attribute_express Source.vi`. Quick Drop allows you to define shortcuts which accelerates the programming even more.
Note: Quick Drop only searches among the VIs that are already in LabVIEW palette. Use the LabVIEW palette editing (Menu Tools -> Advanced -> Edit Palette Set) and switch synchronization with directory content if necessary.

Using **Palettes** or **Quick Drop** will immediately invoke Configuration panel linked to Express VI. To disable this, go to **Menu -> Tools ->Options -> Block Diagram** and uncheck the checkbox **Configure Express VIs immediately**.

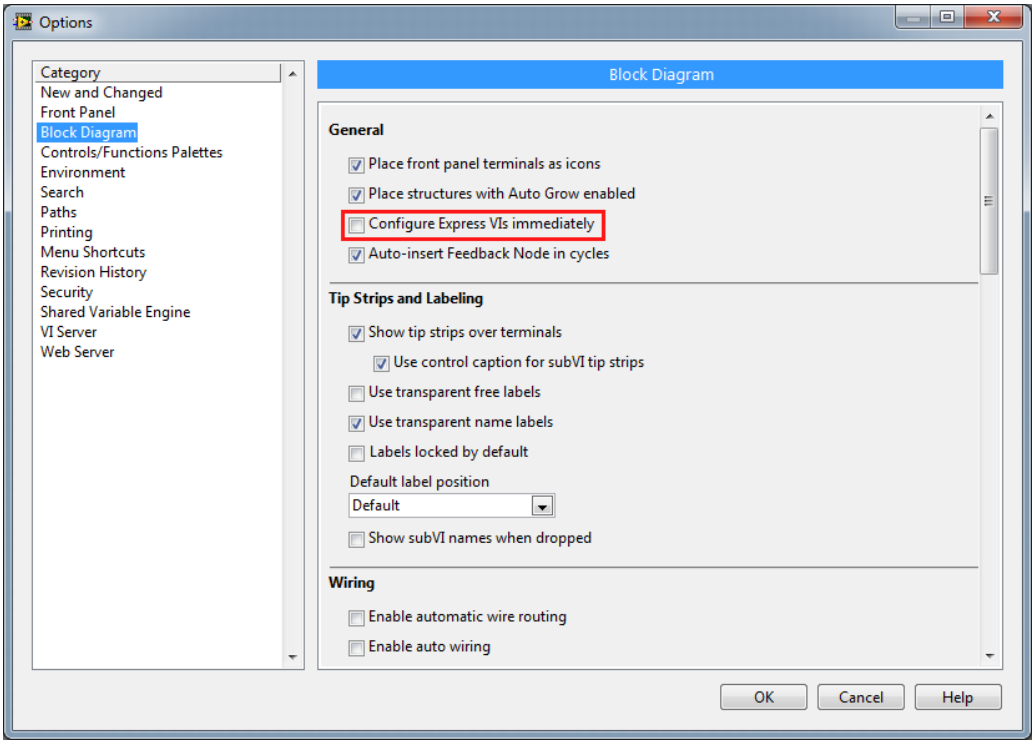


Fig. 4-4: Configure Express VIs immediately settings.

LabVIEW places the Express VI instance on your Block Diagram in full view:

Fig. 4-5: Express VI instance in full view.

You can explore available terminals by dragging red-marked arrow:

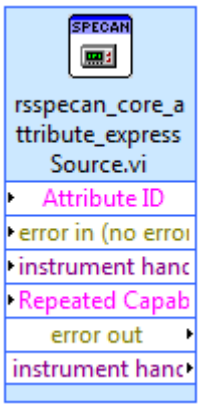
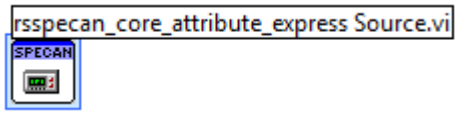


Fig. 4-6: Express VI instance with expanded terminals.

Right-click context menu item **View As Icon** switches to standard Icon view:



After placing / reconfiguring the Express VI instance you can simply make copies of it in your Block Diagram by **CTRL+C/V** or **CTRL+Dragging** (making a copy) or **CTRL +SHIFT+Dragging** (making a copy with vertical or horizontal alignment). The Express VI instance configuration is copied as well. Configuration panel also generates quick help content. You can view it in Context Help window (shortcut **CTRL+H**). As an example see figure 4-14.

4.3 Express VI Configuration panel

Double-clicking on an Express VI instance or selecting **Context-menu -> Properties** brings up the Configuration panel. See how to speed up the loading of the Express VI Configuration panel here: chapter 6.3.2, "Speed up the loading of Express VI Configuration panel", on page 40.

Below, the Configuration panel is described in detail:

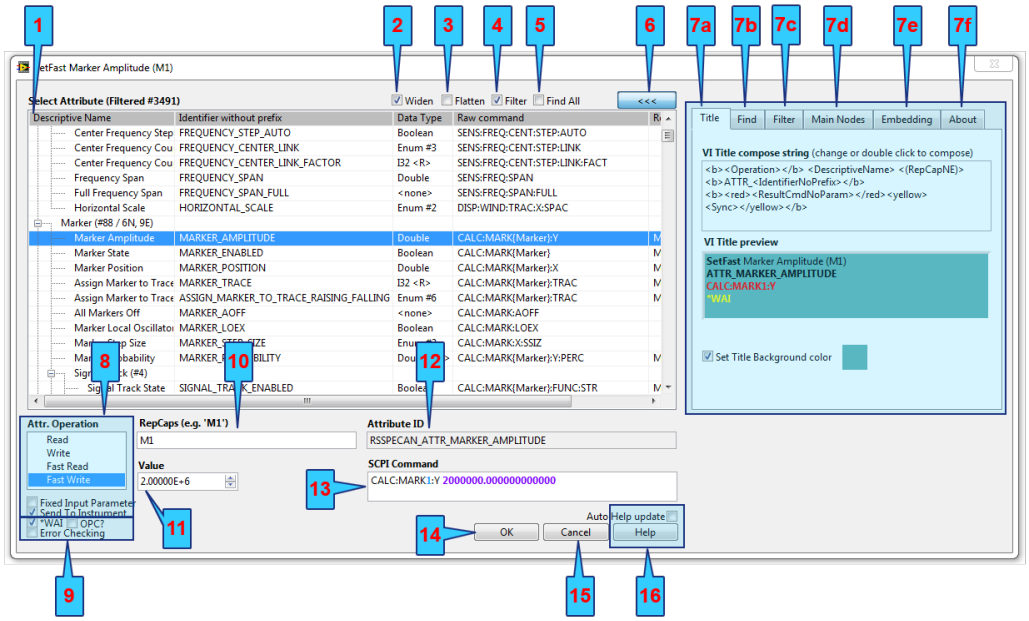


Fig. 4-7: Express VI Configuration panel.

- **1 - Select Attribute** control: This control represents the tree-structure of driver attributes, same as you can find in `rsspecan_attr.chm`. Based on the selected attribute, the availability of other controls will change accordingly (e.g. read/write access, RepCaps, Input data type ...).
  Right-click context menu allows for opening/closing all elements and defining the content up to 5 columns from following attribute values: *Descriptive name, Identifier, Identifier without prefix, Raw Command, Raw Command without { } portions, Data Type, Access, RepCap Definition, R/W Callbacks*.
  Clicking on the column header allows the ascending/descending sorting by that parameter. Sorted column is displayed in bold font with ascending or descending symbol at the beginning. Sorting works best with flatten structure of the attributes tree.

- **2 - Widen the Select Attribute** control: Widens the Select Attribute control, so other columns are better visible. When unchecked, the text box with attribute help text is also available.

- **3 - Flatten the Attribute tree**: If checked, all Select Attribute control nodes are removed and the entire content is shown as a list.

- **4 - Filter attributes**: If ON, only the attributes fulfilling the **Filter** criteria (defined in **7c**) and **Main Nodes** (defined in **7d**) will be visible.

- **5 - Find All**: If ON, only the attributes fulfilling the **Find** criteria (defined in **7b**) will be visible. You can combine **Filter** and **Find** criteria to perform logical AND operation between them.

- **6 - Window size** button: You can choose two different sizes of Configuration panel. The smaller one is a default recommenced size for Express VI Configuration panel, so it fits to 1280x1024 screen resolution. Adjust the **Window size** and the **Widen** (**2**) controls to achieve the desired layout.

- **7a .. 7f**: Refer to chapter 4.3.2, "7a .. 7f tab control tabs", on page 24.

- **8 - Attribute operation**: Refer to chapter 4.3.1, "8 - Attribute operation control", on page 22.

- **9 - Post operation**: Depending on selected **Attribute operation**, *WAI / *OPC? synchronization and Error Checking are available. Refer to chapter 4.3.1, "8 - Attribute operation control", on page 22. For details on different synchronization methods refer to chapter 6.2, "Synchronization methods", on page 37.

- **10 - RepCaps string** control: This string contains values for repeated capabilities defined in SCPI command by information in curly brackets. Repeated capabilities are case sensitive, separated by comma (if there are more than one), no spaces are allowed. Actual values depend on attribute definition and can be found in attribute help under **Supported Repeated Capabilities**. If the selected attribute has 1 or more Repeated capabilities defined, this control is enabled for editing. Invalid repeated capabilities string will be marked with red label and will disable **OK** button (**14**). Repeated capabilities string is allowed to be empty for standard Read/Write operation (if no variable RepCaps are selected), for all others it must be valid. Editing options besides free writing:
  - Right-click context menu **Fill with 1st options** will set the first valid Repeated capabilities string.
  - Double-click on empty RepCaps string control will do the same as **Fill with 1st options**, or in case the attribute has no definition for Repeated capabilities, RepCaps string control will be cleared.
  - Double-click on valid repeated capability text portion will pop-up the listbox with all available options. There you can use left/right arrow to switch between the RepCaps (if there is more than one) or up/down arrow to select a value.

- **11 - Input value**: Input value for **Write** operations. Type of the control depends on attribute **Data type**. If range checking is defined to the selected attribute, you can see the allowed range in its label. In case of **Read** operation this value is disabled and ignored. For data type **Enum** listbox item names depend on settings defined in **7e - Enum Embedding**.

- **12 - Attribute ID**: Selected attribute value indicator.

- **13 - SCPI Command** indicator: This indicator is showing a command that with the current settings will be sent to the instrument. Black text is fixed content, light blue text shows Repeated capabilities portions, red text shows Repeated capabilities portions when the RepCaps string is not valid. Purple portions are the parameters. In case of the **Read** operation, parameter is the questionmark.

- **14 - OK button**: This button finishes the configuration and the Express VI instance is modified according the settings. In case of configuration error or conflict, **OK button** is disabled. Hovering over it with the mouse will bring up the error description text.

- **15 - Cancel button**: Discards all the configuration changes and closes the configuration panel.

- **16 - Help button**: Opens the `rsspecan_attr.chm` file page with the current attribute. Optionally, you can have this automatically done with every change of attribute by checking **Auto Help update** checkbox.

4.3.1 8 - Attribute operation control

Attribute operation defines what is to be done with the selected attribute. In principle, there are only 2 options: Read or Write. Other operations are derived from those two. Attribute operation also affects the Express VI instance icon. *Note for Express VI icon: By default, all Hi-level functions in the driver have the icon of **Standard Write** without the blue arrow.*

Common features of Express VI icon: **Standard** operations have a blue strip on Icon top, **Fast** operations have red/orange strips. A small pink arrow on the left signals that you can connect the **Direct cmd in** string that will be placed before composed string:

Fig. 4-8: Standard Write and Fast Write icons.

If Error Checking is ON, icon has a bubble with **'E?'** in left bottom corner. `*OPC?` synchronisation is shown as a small red vertical strip on the right edge. `*WAI` synchronisation is signaled as a longer blue vertical strip on the right edge:

Fig. 4-9: Fast Write icons: with Error Checking / with OPC? synchronisation / with *WAI synchronisation.

Listed below are all possible attribute operations with their icons. Stated in brackets are the names of Operation variable used in VI Title composing (see chapter 4.3.2.1, "7a - Title composer tab", on page 24) :

- **Read** (Get) - Standard driver read method. All Read functions in driver are programmed using this operation.

Fig. 4-10: Standard Read Icon

- **Write** (Set) - Standard driver write method. All write functions in driver are programmed using this operation.

- **Fast Read** (GetFast) - Fast read operation, command is prepared during configuration. Run-time action is limited to writing prepared string to instrument, reading the response from instrument and converting the response to defined attribute type. You can use **Error Checking** with this operation.

- **Fast Write** (SetFast/SetFix) with **Send To Instrument checked**. Fast write operation to the instrument. If you don't need to change the attribute input parameter during run-time, check **Fixed input parameter** checkbox. Available post-operations: ***WAI**, **OPC?** and **Error Checking**.

- **Fast Compose** (BuildFast/BuildFix) with **Send To Instrument unchecked**. This operation doesn't communicate with the instrument. It only composes the command and returns composed string in **Direct cmd out**, which can be connected to the next Express VI instance **Direct cmd in** input. This way you can compose the list of parameters which will then be sent to the instrument all at once. If you don't need to change the attribute input parameter during run-time, check **Fixed input parameter** checkbox. Available post-operations: ***WAI**.

4.3.2 7a .. 7f tab control tabs

4.3.2.1 7a - Title composer tab

This tab defines the composer string based on which the result Express VI instance title will be composed. This composer string is unique for every Express VI instance. It is a multi-line string that contains fixed portions, variables (e.g. <DescriptiveName>) and formatting pair-tags (e.g. **BoldText**). You can double-click on this control to use the composer window or modify it directly. **VI title preview** shows how the result VI title will look like for the current configuration. Below is the list of available variables and formatting pair-tags:

Table 4-1: Title compose string variables

Variable Name (case sensitive)	***Description***
DescriptiveName	Descriptive name of attribute e.g. **Averaging State**
Identifier	Whole Attribute identifier e.g. **RSSPE-CAN_ATTR_AVG_STATE**
IdentifierNoPrefix	Attribute identifier without <driver_name>_ATTR e.g. **AVG_STATE**
Operation	Attribute operation: **Get / Set / GetFast / SetFast / BuildFast / SetFix / BuildFix**. Refer to chapter 4.3.1, "8 - Attribute operation control", on page 22.
DataType	Attribute data type: **I32 / Double / Boolean / String / Enum / <none>**. If there is a range defined for this value, you see **<R>** at the end. Enum types have number of items shown at the end (e.g. #6) To obtain raw data type (e.g. **Double**) use **DataTypeRaw**)
DataTypeRaw	Attribute data type: **I32 / Double / Boolean / String / Enum / <none>**
Value	Exact command parameter value that is sent to the instrument e.g. **110000000.000000000000**
ValueHRform	Command value that is sent to the instrument but in shortened form e.g. **110M**
ValueIFfixed	Same as **Value**, if it cannot be changed during run-time (checkbox Fixed Input Parameter is checked). Otherwise empty string
ValueHRformIFfixed	Same as **ValueHRform**, if it cannot be changed during run-time (checkbox Fixed Input Parameter is checked). Otherwise empty string
RepCap	Repeated capabilities string e.g. **TR1**
(RepCapNE)	Repeated capabilities string if it's non-empty. Otherwise empty string
RawCmd	Command directly taken from Attribute definition e.g. **CALC:MARK{Marker}**

ResultCmdNoParam	Result command that is sent to the instrument without parameter e.g. **CALC:MARK1**
ResultCmd	Result command that is sent to the instrument e.g. **CALC:MARK1 OFF**
ReadWrite	Read / Write access: **R / W / R/W**
Sync	Depending on ***WAI** and **OPC?** sychronisation check boxes, this variable can have values **'<empty>'**, **'*WAI'**, **'*OPC?'**, **'*WAI;*OPC?'**

Table 4-2: Title compose string formatting tags

Tag name (case sensitive)	***Description***
Pair tags **, **	Bold and italic text e.g.**This is bold plus italic text and this is normal text**
Pair tags for colors **<red>, <blue>, <green>, <yellow>, <pink>, <purple>, <orange>, <white>**	Applies a color to a text between tags

4.3.2.2 7b - Find tab

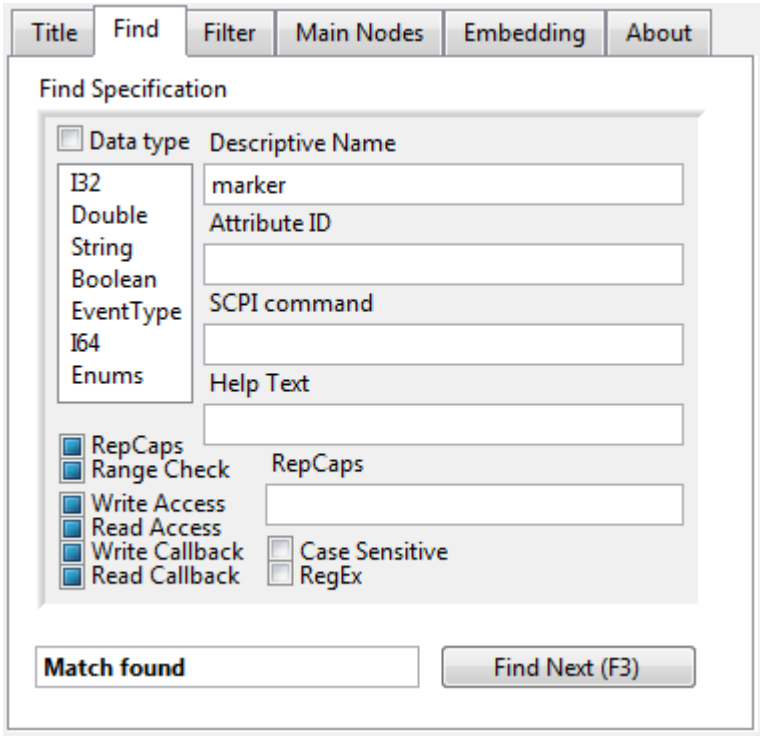


Fig. 4-11: Attributes Find tab.

Find tab allows you to find an attribute based on **Find Specification** control. Searching of the next item fulfilling the criteria can be done with **Find Next** button or **F3**, or **ENTER** button when focused on one of the string fields. You can see all positive search results in Select Attribute control when checking **5 - Find All** checkbox (figure 4-7). Use **CTRL+F** to quickly focus on **Descriptive name** field.

String fields **Descriptive Name**, **Attribute ID**, **SCPI command**, **Help Text** and **RepCaps** allow for full text searching in attribute fields. All non-empty strings must fulfill the attribute specification, otherwise they will not be positively matched.

If **Case Sensitive** checkbox is checked, all non-empty strings are evaluated with case sensitivity.

If **RegEx** checkbox is checked, all non-empty strings are considered LabVIEW regular expressions.

Data type checkbox switches search based on attribute value data type. Multiple items selection is possible with **CTRL** or **SHIFT**.

3-state checkbox RepCaps - search based on the attribute property Repeated Capabilities. If checked, only the attributes with at least one RepCap defined are shown.

3-state checkbox RangeCheck - search based on the attribute property of value range checking. If checked, only the attributes that have a defined range for their value will be shown.

3-state check boxes Write Access / Read Access - search based on attribute property of access. If checked, only the attributes which are writable / readable will be shown.

3-state checkbox Write Callback / Read Callback - search based on attribute property of special function for writing or reading. Some attributes cannot use standard value data types of I32 / Double / Boolean / String, therefore they need special function - called **Callback**. As a consequence, **Fast Write/Read** operations cannot be used with such attributes. If this checkbox is checked, only the attributes for which Write/Read callback is defined will be shown.

4.3.2.3 7c - Filter tab

Filter tab allows to only see the attributes in Select Attribute control that fulfill the **Filter Specification**. Main filter switch is the control **4 - Filter** (figure 4-7). **Filter specification** control is the same as in **7b - Find Tab**.

4.3.2.4 7d - Main Nodes tab

By selecting only a portion of Main Nodes you can select only branches of Attributes tree that are of interest to you. The changes only have an effect when the control **4 - Filter attributes** is checked. By default, all Main Nodes are selected. Multiple items selection is possible with **CTRL** or **SHIFT**.

4.3.2.5 7e - Embedding tab

This tab contains the setting for Express VI instance embedment into parent VI.

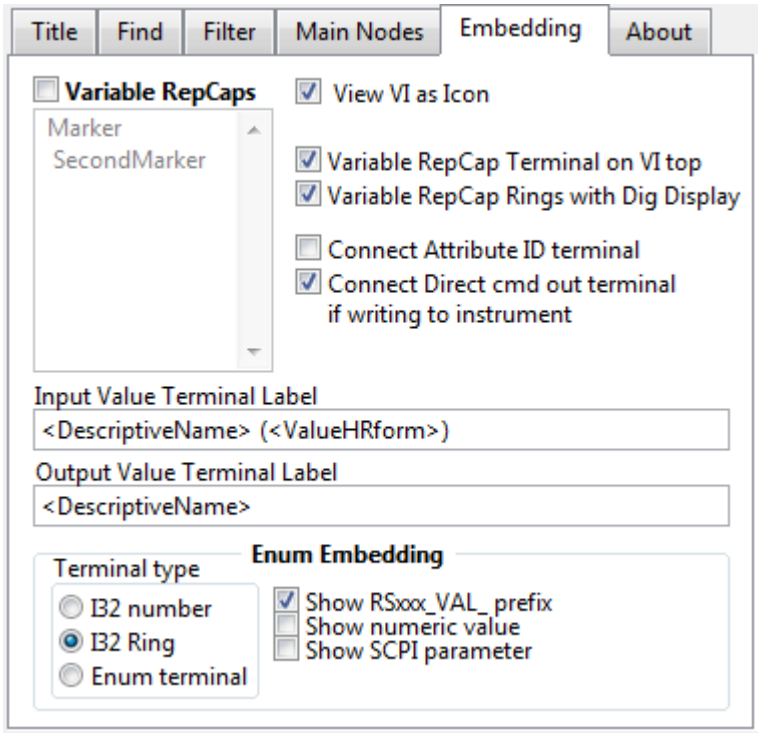


Fig. 4-12: Embedding settings tab.

Embedding settings explanation:

Variable RepCaps: If checked, you can choose 1 or more Repeated capabilities (e.g. Trace, Marker etc.) to be variable during the run-time. RepCaps string in this case must contain a valid Repeated capabilities string (even for standard Read/Write operations) which will be taken as a default value. Express VI instance will contain additional Ring input(s) for Repeated Capabilities with default values taken from RepCaps string.

View VI as Icon: With this setting you can define whether to embed the Express VI instance as an icon. This checkbox is a 3-state control, therefore you can also define not to change the previous settings.

Variable RepCap Terminal on VI Top: With this checkbox you can define whether to connect variable RepCaps terminals on Express VI instance connector top, or bottom.

Variable RepCap Rings with Dig Display: Sets the digital displays on variable RepCaps Ring controls.

Connect Attribute ID terminal: This setting has only effect for Standard Read/Write operations. For Fast Read/Write operations the Attribute ID terminal is never connected, because it cannot be changed during the run-time.

Connect Direct cmd out terminal if writing to instrument: If you chose the Attribute operation that sends the data to instrument (command or query), you can still access the command through this terminal by checking this checkbox (e.g. if you want to log the communication with your instrument).

Input Value Terminal Label: This string control can use the same variables as Title compose string (table 4-1) without rich-text capabilities. Double click to compose the settings. The result name will be used as a label for variable input value terminal.

Output Value Terminal Label: This string control can use the same variables as Title compose string (table 4-1) without rich-text capabilities. Double click to compose the settings. The result name will be used as a label for output value terminal.

Enum Embedding - Terminal Type: You can decide how to embed Enum Data types: as I32 number, I32 Ring or Enum terminal. This settings is common for input and output value terminals.

Enum Embedding - Show RSxxx_VAL_ prefix: If you use embedding of Enum Data Types as I32 Ring or Enum, you can decide whether to have item names with RSxxx_VAL_ prefixes. Sometimes the names are too long, so cutting this part out makes the items more compact.

Enum Embedding - Show numeric value: If checked, the item names start with their integer value in round brackets.

Enum Embedding - Show SCPI parameter: If checked, the item names end with actual SCPI parameter in square brackets. Changing **Show RSxxx_VAL_ prefix**, **Show numeric value** or **Show SCPI parameter** when you have the attribute with Enum Data Type selected will immediately alter **11 - Input Value** Ring items. This way you can see how item names will look in your code.

As an embedding example, open the attached file `LabVIEW_programs\Example_EmbeddingSetTrace.vi`. It shows the Express VI instance with the attribute `RSSPECAN_ATTR_TRACE_TYPE` and the following Embedding options:

Control Name	Value
Variable RepCap	True
RepCaps string	TR1
Variable RepCap Terminal on VI top	False
Variable RepCap Rings with Dig Display	True
View VI as Icon	True
Input Value Terminal Label	<Descriptive Name> <DataTypeRaw> (<Value>)
Enum Embedding - Terminal Type	Ring
Enum Embedding - Show RSxxx_VAL_ prefix	False
Enum Embedding - Show SCPI parameter	True

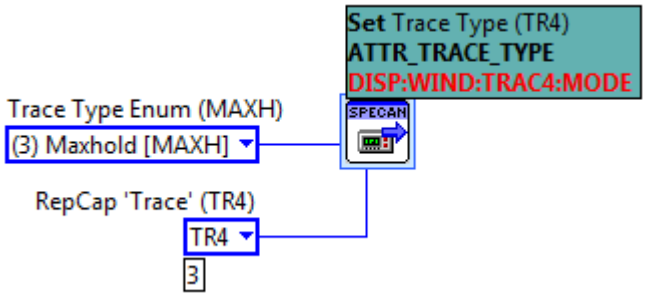


Fig. 4-13: Example of an Embedded Express VI instance.

Open the context help (**CTRL+H**) and hover the mouse over the Express VI to see its configuration and short help for the selected attribute:

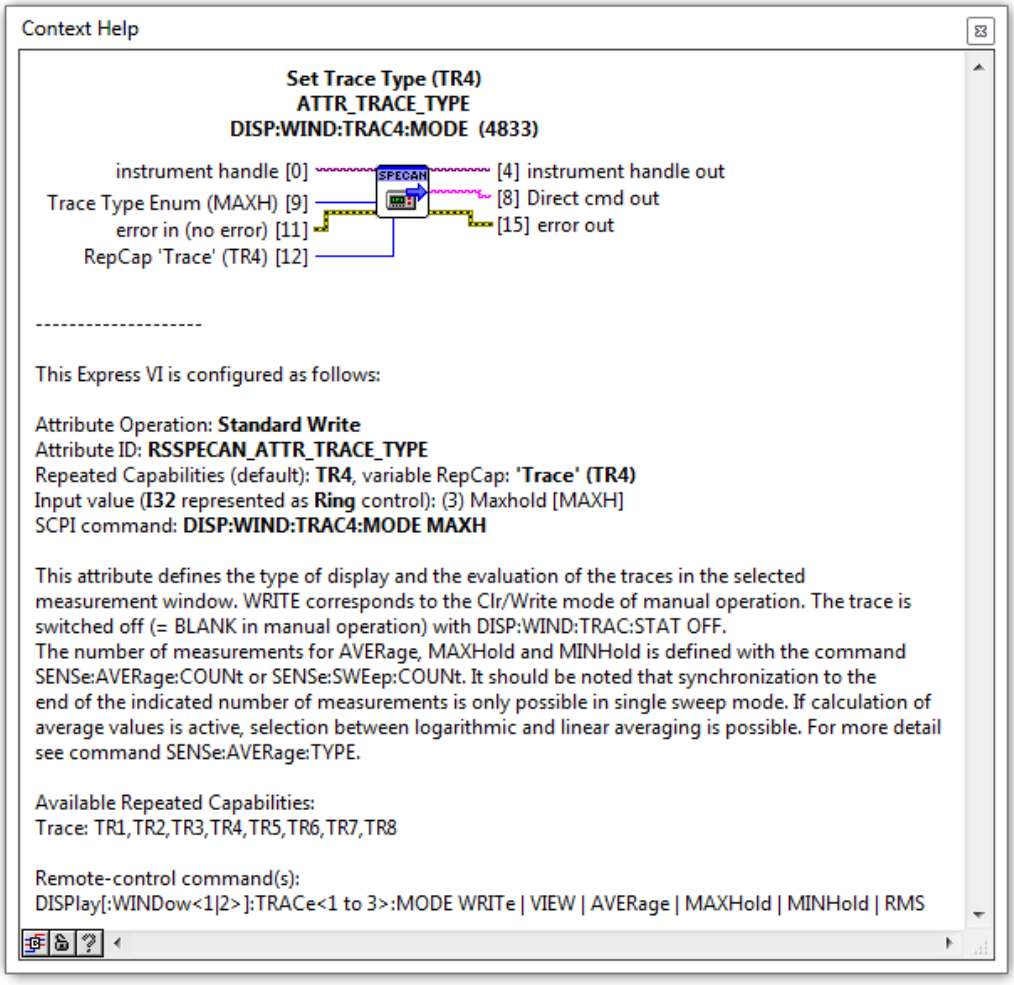


Fig. 4-14: Configuration of the Express VI displayed in Context Help window.

4.4 Example of LabVIEW Express VI code

Attached file: `LabVIEW_programs\Example_Express.vi`

This code opens the VISA session to the analyzer, configures the analyzer to single sweep with 10 maxhold and minhold sweeps (you can add more traces). Then, it reads X and Y traces and displays them in XY graph. Notice the different approaches of communication with the instrument:

Building the SCPI command string with several **Fast Compose** (BuildFast / BuildFix) operations and at the end use **Fast Write** (SetFast) operation. As a result, only the last VI communicates with the instrument. This is the fastest way how to set up you instrument. The SCPI command string that has been built and sent is shown in the string control **Direct cmd out 1**. In our case the SCPI string is
`SYST:DISP:UPD OFF;:INIT:CONT OFF;:SENS:SWE:COUN 3;*WAI`

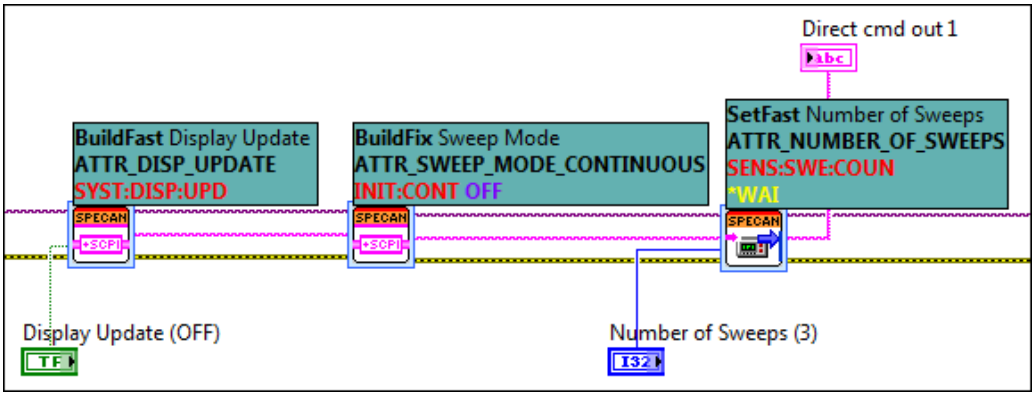


Fig. 4-15: SCPI command string building with two Fast Compose operations, and one Fast Write operation at the end.

Sending the commands with the driver standard **Write** (Set) operation. Commands are sent directly and **Error Checking** is performed after every operation. There is no difference between standard **Write** operation Express VIs and using the driver VIs, because the entire driver is programmed with Express VIs configured to standard **Write** or standard **Read** operations.

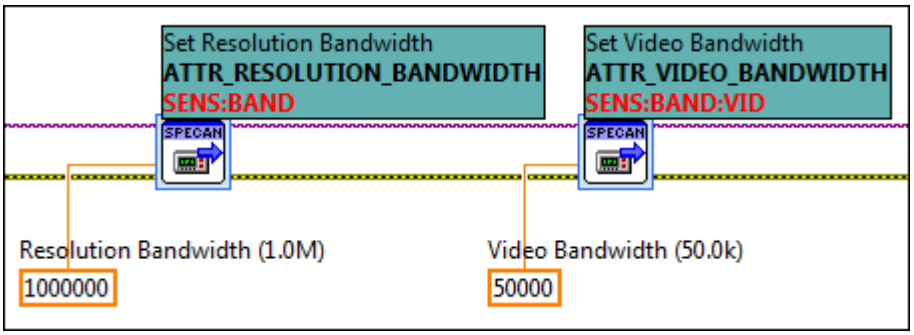


Fig. 4-16: Standard Write VIs that you can find in all driver functions.

Similarly to the previous part SCPI command string is built by using **Fast Compose** (BuildFast/BuildFix), and the end using **Fast Read** (GetFast) to retrieve a response

from the instrument. As you can see SCPI command string can be built by using different methods, in our case with shift register over the for loop creating settings for multiple traces. Compared to sending separate command for every attribute this approach takes only a fraction of that time. In this example, the last Express VI sending SCPI command string also checks for instrument errors. This way you can isolate groups of SCPI commands that have caused an error. The following string (also available in **Direct cmd out 2**) is sent to the instrument:

```
DISP:WIND:TRAC1:MODE MAXH;:DISP:WIND:TRAC2:MODE MINH;:SENS:FREQ:
CENT 254000000.000000000000;:SENS:FREQ:CENT:STEP:LINK RBW;:SENS:
FREQ:SPAN 12000000.000000000000;:SENS:FREQ:CENT:STEP:LINK?
```

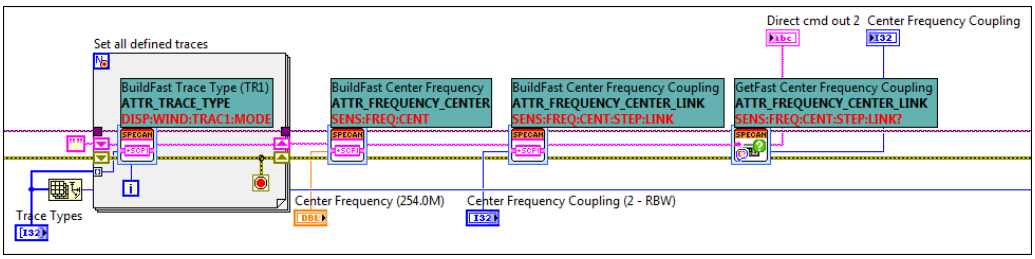


Fig. 4-17: SCPI command string building with several Fast Compose operations, and one Fast Read operation at the end, together with Error Checking.

5 Performance comparison

Attached files:

- `LabVIEW_programs\Performance_comparison_Standard_Fast_Raw.vi`
- `Scripts\Spyder_PlainVISA_WriteRead.py`
- `VS_programs\PlainVISA_WriteRead\PlainVISA_WriteRead.sln`

This chapter contains comparison of performances achieved by using different approaches in LabVIEW and in other programming environments. The following test steps were evaluated:

- ***IDN? loop** time measurement - 10000 cycles of sending `*IDN?` query and reading the response from instrument. This task is not performed when using the LabVIEW driver, since this feature is not available.
- **Configuration** time measurement - setting 8 different parameters to the instrument.
- **Measurement** - 10000 cycles of triggering the short sweep (50µs) in zero span, waiting for the sweep to finish, reading the RMS marker, reading marker X and Y coordinates.

All used programs and scripts are available in attachment of this Application Note. All measurements are performed using VXI-11 and HiSLIP protocols. The following approaches were chosen for comparison:

- LabVIEW driver Standard configuration and measurement functions.
- LabVIEW driver Express VIs configured to Standard operations.
- LabVIEW driver Express VIs configured to Fast operations.
- LabVIEW raw VISA write/read.
- Visual Studio 2012 project in C# with `visa32.dll`.
- Spyder Python script with raw VISA communication using PyVisa component.

Test setup:

- Dell Optiplex 7010, I7-3770 3.40GHz, OS Win7 64-bit, 16GB RAM
- Network adapter: Intel PRO/100Mbit , LAN switch 100MBit
- Instrument: Rohde & Schwarz FSW26, firmware 2.00
- LabVIEW 2010 64-bit
- LabVIEW driver rsspecan 3.1.0, 07/2014
- Visual Studio Professional 2012 Version 11.0.51106.01 Update 1, .NET framework 4.5.50709
- Spyder 2.2.5 with Python 2.7.6 32-bit

5.1 Results using VXI-11 protocol

Table 5-1: Result table for VXI-11 protocol. EC OFF means that Error Checking was switched OFF.

	*IDN? loop time in seconds	Configuration time in seconds	Measurement time in seconds
LabVIEW driver Standard configuration	N.A.	0.07 **0.04** (EC OFF)	64.07 **45.22** (EC OFF)
LabVIEW driver Express VIs Standard operations	N.A.	0.04 **0.03** (EC OFF)	49.73 **30.21** (EC OFF)
LabVIEW driver Express VIs Fast operations	N.A.	0.03	24.97
LabVIEW raw VISA write/read	9.59	0.04	24.26
Visual Studio 2012 C#	9.26	0.03	23.51
Spyder Python with PyVISA	10.51	0.04	27.04

Below, the last column **Measurement time in seconds** represented in graph, sorted by speed from the slowest to the fastest:

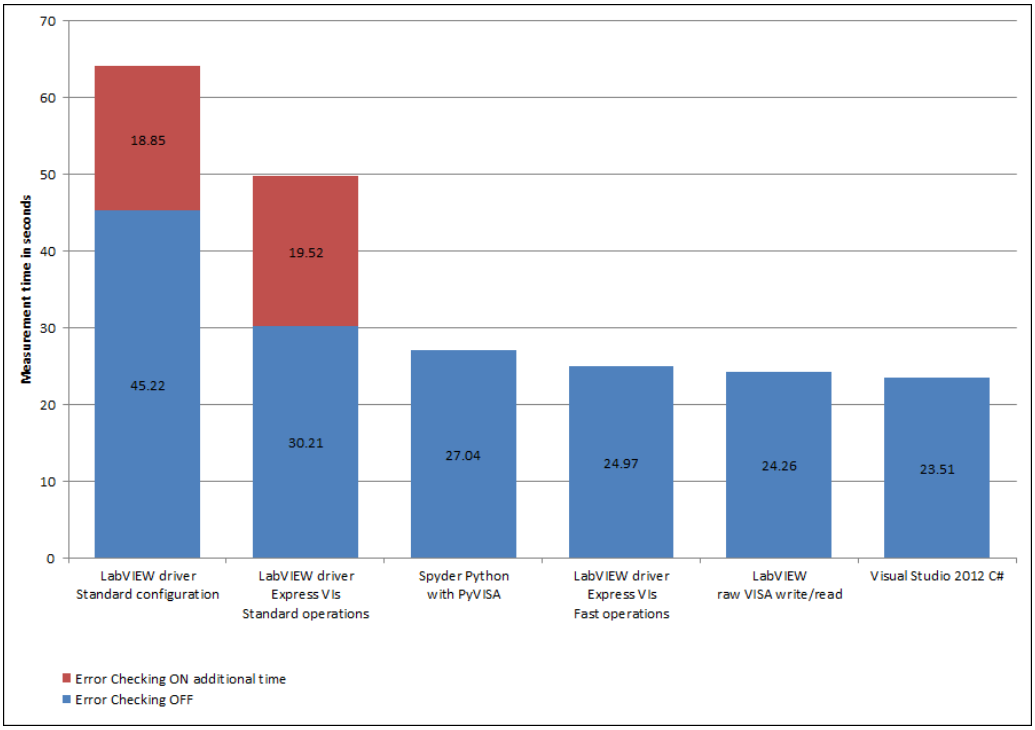


Fig. 5-1: VXI-11 duration time results of the Measurement task.

5.2 Results using HiSLIP protocol

Table 5-2: Result table for HiSLIP protocol. EC OFF means that Error Checking was switched OFF.

	*IDN? loop time in seconds	Configuration time in seconds	Measurement time in seconds
LabVIEW driver Standard configuration	N.A.	0.07 **0.06** (EC OFF)	41.27 **31.65** (EC OFF)
LabVIEW driver Express VIs Standard operations	N.A.	0.03 **0.03** (EC OFF)	31.47 **21.44** (EC OFF)
LabVIEW driver Express VIs Fast operations	N.A.	0.03	20.92
LabVIEW raw VISA write/read	2.03	0.03	20.75
Visual Studio 2012 C#	2.03	0.02	20.45
Spyder Python with PyVISA	2.48	0.07	22.62

Below, the last column **Measurement time in seconds** represented in graph, sorted by speed from the slowest to the fastest:

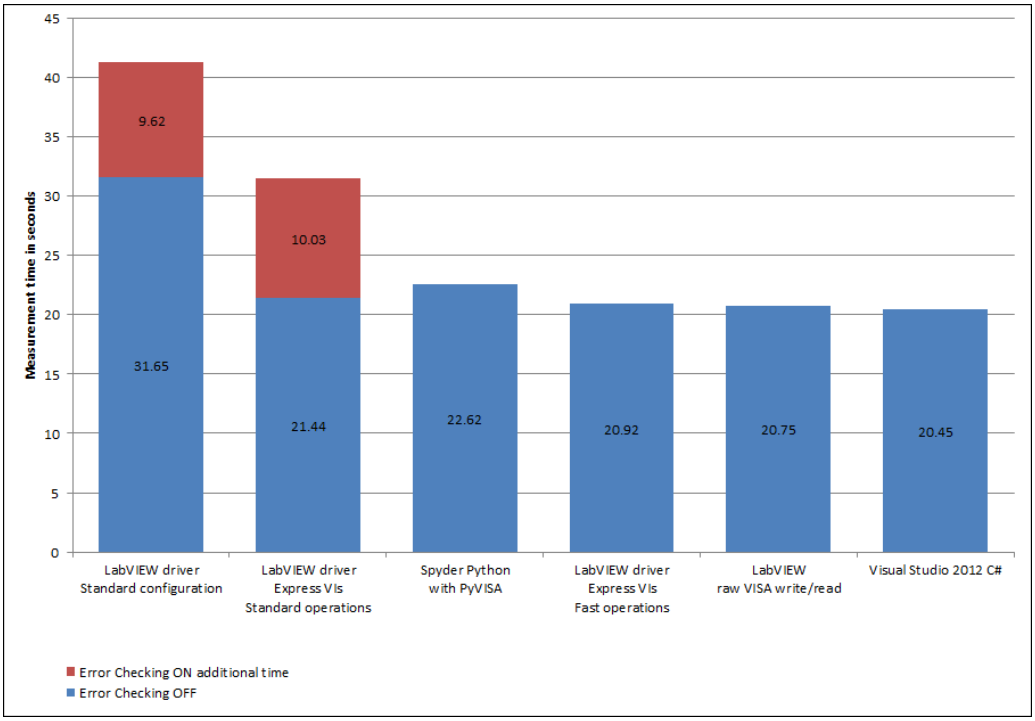


Fig. 5-2: HiSLIP duration time results of Measurement task.

5.3 Conclusion

When using the LabVIEW driver, switching OFF the Error Checking can significantly reduce the measurement time (over 30% in VXI-11, over 20% in HiSLIP in our measurement task). The absolute time spent on one error checking task is in our setup cca. 1.9ms (VXI-11) resp. 1ms (HiSLIP). The downside of Error Checking switched OFF is, that you cannot react on an error occurred in the instrument which can lead to although fast, but inaccurate or invalid measurement results. The best approach is to do the 'Smart' Error Checking:

- Keep Error Checking ON for sections of your program that are not performed in multiple loops.
- Keep Error Checking ON for multiple measurement loops where the instrument measurement times are relatively long (long sweep times, averaging results, etc.). The longer measurement time you have, the less significant is the Error Checking time. By sweep times in range of seconds the Error Checking time is negligible.
- Switch Error Checking OFF for critical portions of your measurement with multiple loops, do the error check before and right after them.

The difference between the **LabVIEW driver Standard configuration** (1st column) and **LabVIEW driver Express VIs standard operations** (2nd column) is caused by `Initiate.vi` which is doing additional operations besides sending `INIT;*WAI` SCPI command string (clearing the instrument error queue, temporarily changing the session based measurement timeout). When you use Express VI with `RSSPECAN_ATTR_INIT` instead, you will achieve the same measurement times in both cases.

In HiSLIP **LabVIEW driver Express VIs Standard operations** with Error Checking OFF are only cca. 5% slower than **Visual Studio 2012 C#**. In VXI-11 this difference grows to cca. 28%.

It is safe to say that when it comes to communicating with instruments over VISA, LabVIEW in raw write/read mode is as fast as other programming languages. When using the Rohde & Schwarz LabVIEW drivers, **Express VI** with **Fast Operations** can bring the performance very close to raw write/read mode.

Of course the communication with instruments is only one part of measurement application, comparing the overall performance including user interfaces, graph and data handling is the topic beyond the scope of this Application Note. It is also fair to mention, that the performance comparison was done on relatively powerful PC. The results can differ depending on the used computer. Therefore we encourage the user to do the comparisons on his setup using attached programs and scripts.

6 Tips and Tricks

6.1 Building Executables

All Rohde & Schwarz attribute-based drivers use standard functions for writing/reading the following attribute data types: I32, Double, String, Boolean, Enum (treated internally as I32). In case that attribute parameter cannot be formatted using these standard types, the drivers use functions called `Callbacks`. Usually, a pair of Callbacks is defined for both writing and reading operations. All of these callbacks are stored in the driver structure `Private\callbacks` folder. When running in development environment, the driver dynamically calls the Callback if an attribute requires it. But if you create an executable with your Top-level VI, none of the Callbacks are in its dependency list, therefore they will not be included into your executable application. As a consequence, if you run your executable, the first use of attribute requiring a Callback will report the error `-1074003958 (0xBFFC000A)` with following text: `Cannot find dynamically called callback '<callback_name>'. Make sure that all VIs are in correct folders. If deploying, folder structure has to be maintained!`

To solve this error, go to your Project Explorer window Build Specifications and open your desired application (in our case MyApplication) Properties window:

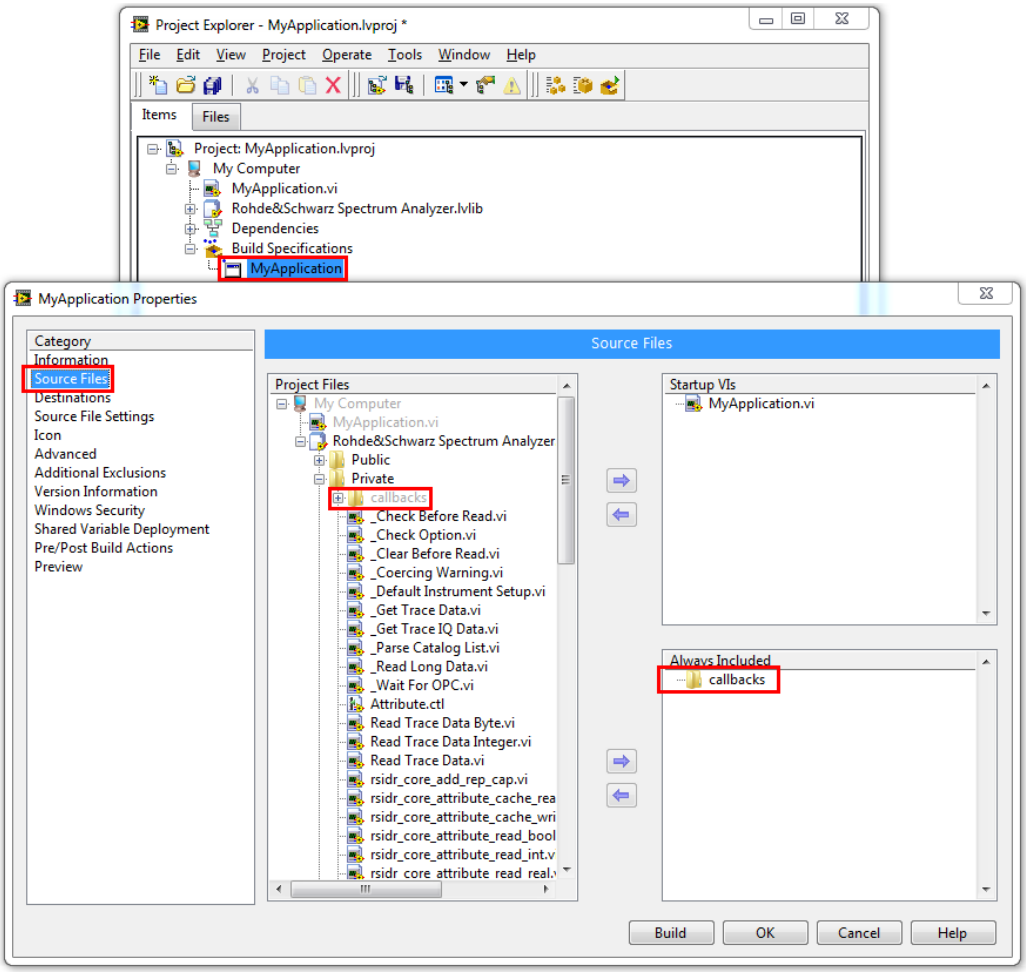


Fig. 6-1: Building an executable requires adding entire callbacks folder to Always Included set of VIs or folders.

Select **Source Files**, in **Project Files** open `Rohde&Schwarz Spectrum Analyzer.lvlib` element and navigate to `Private\callbacks` folder. Add the entire folder to **Always Included** group.

In order to see the `Rohde&Schwarz Spectrum Analyzer.lvlib` in Project Files, you need to have it added to `MyApplication.lvproj`. If your instrument driver is not project-based, add its `_utility\callbacks` folder to `MyApplication.lvproj` in order to access it.

If you are using more than one driver, you will need to repeat this procedure for each of them.

6.2 Synchronization methods

Synchronization of your application and your measurement instruments is a crucial part of every automated measurement task. Programmer always has to know in which state his instrument is when he applies a stimulus or reads the measurement results. Using

instrument locally and observing measurement results visually is in this aspect very different from operating it remotely. While in local operation user's eye serves as a flag to distinguish between already valid result and measurement still in progress, remote control application must rely on instrument's build-in synchronization mechanisms. Although the measurement synchronisation is mostly related to analyzer-class instruments (spectrum analyzers, oscilloscopes, audio analyzers, multimeters), the same principles can be used also for other instrument types.

Keep in mind, that in order for synchronization to be working properly your instrument must be in single sweep / single measurement mode. Below, there are three basic synchronization methods you can use with Rohde & Schwarz instruments.

6.2.1 *WAI command

Attached example: `LabVIEW_programs\Example_synchronization_WAI.vi`

Sending `*WAI` command makes your instrument (not your application !!!) to wait until all previous pending commands are completed. This method doesn't synchronize instrument with your application, it just tells the instrument that you want it to finish all the previously received commands before processing further ones. *Example of use*: Send the following string to you spectrum analyzer (1st part of attached example):
`SYST:DISP:UPD OFF;:INIT:CONT OFF;:SENS:SWE:TIME 3.000000000000;:SENS:SWE:COUN 1;*WAI;:INIT;*WAI;:CALC:MARK1:MAX;:CALC:MARK1:Y?`
Notice that you can send the entire SCPI string together and because of `*WAI` command the instrument will do exactly what you expect: Set up, wait, one sweep, wait for it to finish, set the marker to max and return you the amplitude. *WAI assures that you get the marker value only when the sweep is finished and you're not getting an invalid result or a result from previous sweep. This process also makes your application wait in `VISA Read.vi` and therefore continue only when the measurement was finished. If you want your application to perform other tasks while the sweep is running, you have to send the same SCPI command string without the query at the end and only later query the marker amplitude (2nd part attached example) In this case your application meantime operation must be shorter than the duration of the sweep, otherwise your analyzer waits idle. Not to lose time and find out whether measurement result is already available, you have to use *OPC/STB poll synchronization method.

6.2.2 *OPC? query

Attached example:
`LabVIEW_programs\Example_synchronization_OPCquery.vi`

Querying (write + read) `*OPC?` makes your application to wait until the instrument responds with '1' (or '0' if error occurred) after all previous operations were finished. Attached example shows the same task as in case of `*WAI` but using the `*OPC?` query. You need to set the **VISA timeout** parameter to higher value than the duration of your measurement.

6.2.3 *OPC/STB poll

Attached example:
`LabVIEW_programs\Example_synchronization_OPC&STBpoll.vi`

Instrument's Event status register (**ESR**) provides an event-like (reading the value clears it) information about the instrument status. Its bit 0 (Operation Complete) is set to 1 when all the previous operations have finished. If you set the Event status enable register (**ESE**) bit 0 to True, Operation Complete event will be reported in Status register (**STB**) bit **5-ESB** (see the figure below). The driver sets the **ESE** bit 0 to True in **Intialize.vi**.

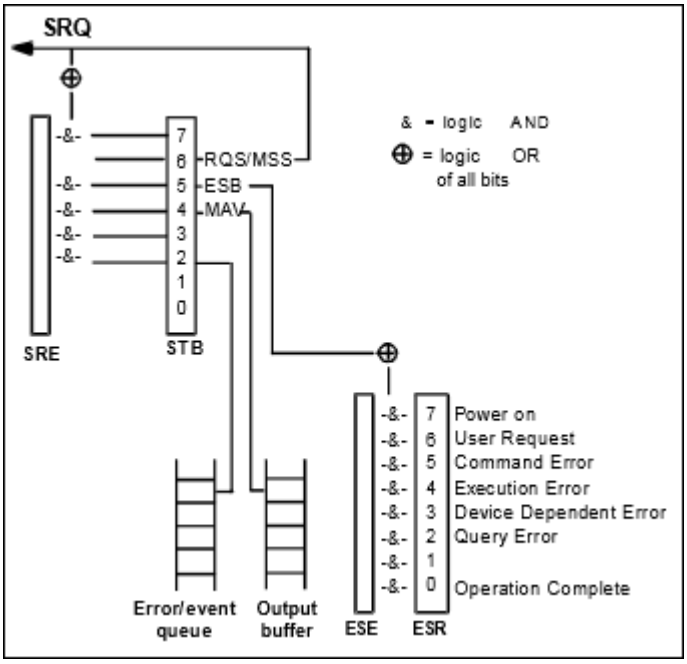


Fig. 6-2: Instrument status reporting system showing ESR, ESE and STB registers.

Sending `*OPC` at the end of the command makes the instrument generating the Operation Complete event in **ESR** after all pending commands have been processed. The occurrence of this event can be then read from **STB** register bit **5-ESB**. The reason for reading the **STB** register instead of direct reading of the ESR register bit 0 is, that VISA provides **viReadSTB** function. This function doesn't query the instrument at all (as opposed to ***ESR?** or ***STB?** queries), but only reads the session internal **STB** register on controller. The **STB** content synchronization between the controller and the instrument is handled by the instrument session automatically, and therefore it optimizes the traffic and the speed on the controller-instrument interface.

The advantage compared to `*OPC?` synchronization method is, that your application can perform other operations while waiting for the instrument to report the operation complete. This method of synchronization is used internally by the driver in function `Wait For OPC.vi`. As shown in the example you need to use the timeout different from **VISA timeout** to prevent a dead-lock. The Express VI doesn't provide this synchronization option.

6.3 Tips when using drivers

In this chapter you will find useful tips when working with Rohde & Schwarz drivers.

6.3.1 Use cross-references in driver documentation

Rohde & Schwarz drivers always come with Microsoft compiled HTML documentation file (e.g. `rsspecan.chm`) that contains description of all VIs also accessible through LabVIEW context help (**CTRL+H** shortcut). The indexed portions are VI names, SCPI commands and attribute names. Therefore, in **Index** tab you can find cross-links between those items. This offers the possibility to search for specific SCPI command and see in which Hi-level function or attribute it is used, which attribute is used in which Hi-level function etc.

6.3.2 Speed up the loading of Express VI Configuration panel

Rohde & Schwarz drivers are compiled in LabVIEW 2010. If you use higher version of LabVIEW, it is the best to perform **Mass Compile** of the entire driver folder before using it. This is especially significant for Express VI Configuration panel, because LabVIEW will never do it automatically and that leads to compilation of this code every time you start Express VI configuration. Mass compiling of a big driver like rsspecan can take some time, but you only need to do it once.

After copying `instr.lib` and `user.lib` driver folders to your LabVIEW folders, start LabVIEW, go to menu **Tools -> Advanced -> Mass Compile**, in **Directory to compile** navigate to `c:\Program Files\National Instruments\LabVIEW 2010\user.lib\_express\rsspecan` and press **Mass Compile** button.

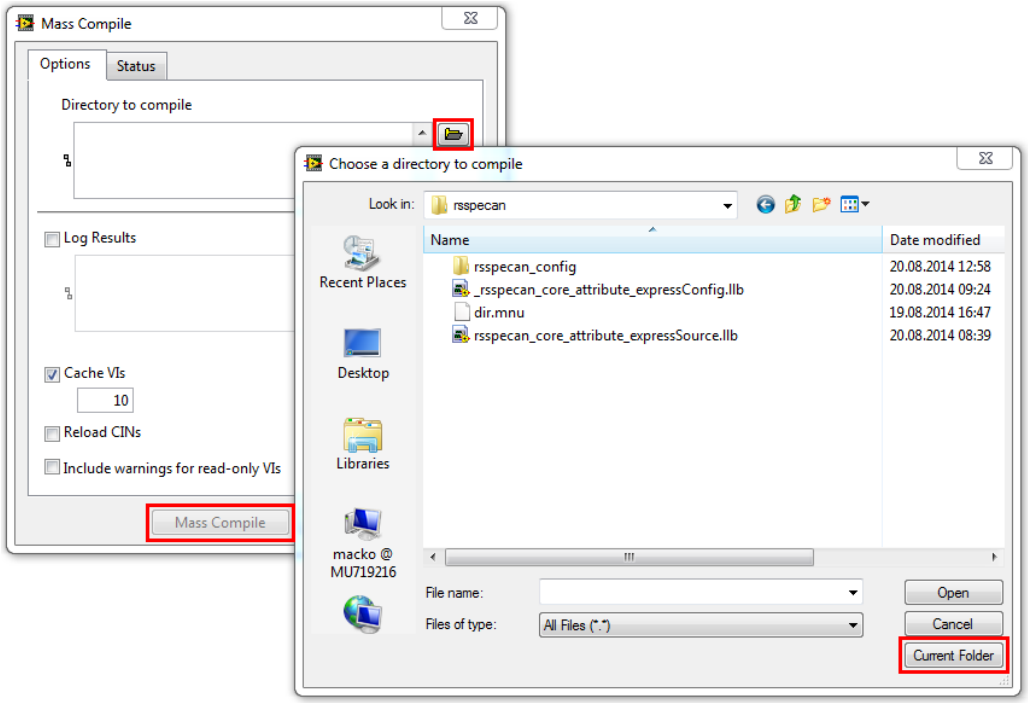


Fig. 6-3: LabVIEW Mass Compilation of Express VI folder.

LabVIEW will compile all VIs in the selected folder and all sub-folders to the actual version. It also compiles all VIs in `llb` libraries. Repeat the same procedure for the folder `c:\Program Files\National Instruments\LabVIEW 2010\instr.lib\Rohde&Schwarz Spectrum Analyzer`.

6.3.3 Use Express VI to find an attribute

The driver Express VI has a searching and filtering capabilities (chapter 4.3.2.2, "7b - Find tab", on page 25) which offer more options than the help file. Usually you know what data type your parameter is (you can also select multiple data types) and part of the descriptive name. If you know a part of the SCPI command you're looking for, entering e.g. `calc:mark:max` in **SCPI command** field, will also positively match attribute with command `CALC:MARK{Marker}:MAX`, because the portions defining RepCaps in SCPI command are ignored during the search.

Configuration panel control **Select Attribute** has customizable columns content with right-click context menu. Use it to display the desired attribute properties.

6.3.4 Analyzer should always be in Single Sweep mode

Always keep your analyzer-class device in single-sweep mode. For that, in the **rsspecan** driver use the attribute `RSSPECAN_ATTR_SWEEP_MODE_CONTINUOUS`, or Hi-level function `Configure Acquisition.vi`.

6.3.5 Initialize and Continue measurement

Starting the measurement on spectrum analyzer is performed with the attribute `RSSPECAN_ATTR_INIT`, or Hi-level function `Initiate.vi`. If you don't want to clear the previous Average/MaxHold/MinHold results, use the attribute `RSSPECAN_ATTR_INIT_CONMEAS` or `Continue.vi`. Both `Initiate.vi` and `Continue.vi` send `*WAI` at the end, which assure that if you send any query immediately after them, the query will be processed only when the measurement sweep is finished. Therefore your query will return a result from the last sweep.

6.3.6 Preventing measurement Timeout

Synchronization method used by the Rohde & Schwarz drivers is *OPC/STB poll. Therefore changing **VISA Timeout** parameter will not help avoiding timeouts for long measurements. Use **Utility** VIs to set/get the STB poll timeout value (chapter 3.1.1.4, "Get Timeout.vi / Set Timeout.vi", on page 12).

6.3.7 Reading and Fetching measurement results

Vis that have **Read** in their names start a new measurement and then read the results. If you don't want to start a new measurement, use **Fetch** VIs. They only read the values obtained by the last measurement.

7 Additional Information

Please send your comments and suggestions regarding this Application Note or Attribute Express VI to:

TM-Applications@rohde-schwarz.com

Using tag “**[1MA228]**” in the mail subject will help us to quickly identify the topic and speed up the response process.

8 Rohde & Schwarz

Rohde & Schwarz is an independent group of companies specializing in electronics. It is a leading supplier of solutions in the fields of test and measurement, broadcasting, radiomonitoring and radiolocation, as well as secure communications. Established more than 80 years ago, Rohde & Schwarz has a global presence and a dedicated service network in over 70 countries. Company headquarters are in Munich, Germany.

Sustainable product design

- Environmental compatibility and eco-footprint
- Energy efficiency and low emissions
- Longevity and optimized total cost of ownership

Certified Quality Management
ISO 9001

Certified Environmental Management
ISO 14001

Regional contact

- Europe, Africa, Middle East
  Phone +49 89 4129 12345
  customersupport@rohde-schwarz.com
- North America
  Phone 1-888-TEST-RSA (1-888-837-8772)
  customer.support@rsa.rohde-schwarz.com
- Latin America
  Phone +1-410-910-7988
  customersupport.la@rohde-schwarz.com
- Asia/Pacific
  Phone +65 65 13 04 88
  customersupport.asia@rohde-schwarz.com
- China
  Phone +86-800-810-8228 / +86-400-650-5896
  customersupport.china@rohde-schwarz.com

Headquarters

Rohde & Schwarz GmbH & Co. KG

Mühldorfstraße 15 | D - 81671 München

+ 49 89 4129 - 0 | Fax + 49 89 4129 – 13777

www.rohde-schwarz.com