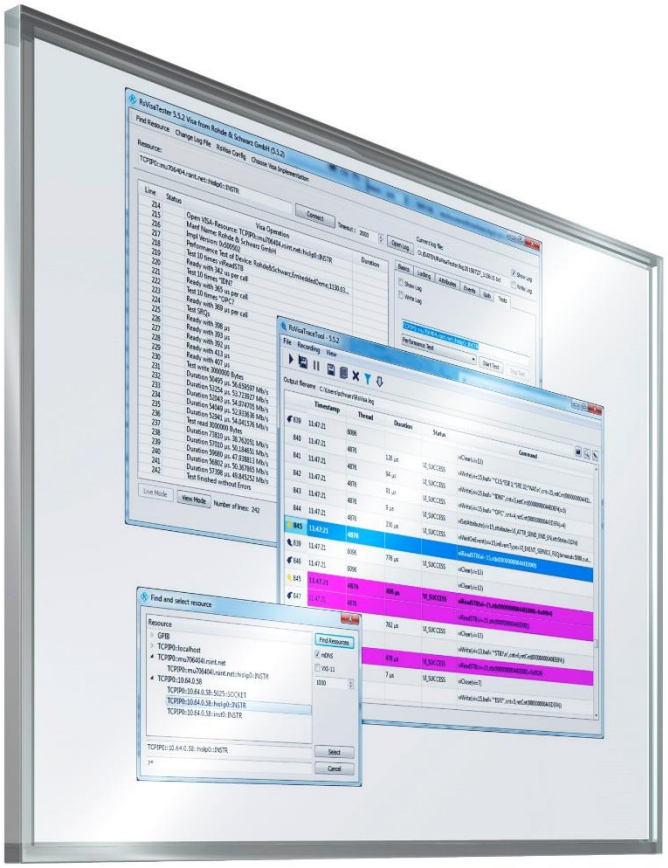
R&S®VISA

User Manual



1700.0232.01 – 03

Test & Measurement

User Manual

PAD-T-M: 3574.3259.02/01.10/CI/1/EN

ROHDE&SCHWARZ

The R&S VISA makes use of the VISA Shared Components by the IVI® Foundation.

IVI Foundation Copyright Notice

Content from the IVI specifications reproduced with permission from the IVI Foundation.

The IVI Foundation and its member companies make no warranty of any kind with regard to this material, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. The IVI Foundation and its member companies shall not be liable for errors contained herein or for incidental or consequential damages in connection with the furnishing, performance, or use of this material.

Muehldorfstr. 15, 81671 Munich, Germany
Phone: +49 89 41 29 - 0
Fax: +49 89 41 29 12 164
E-mail: info@rohde-schwarz.com
Internet: http://www.rohde-schwarz.com/rsvisa

The following abbreviations are used throughout this manual: R&S® VISA is abbreviated as R&S VISA.

1 Contents

1 Introduction

VISA (Virtual Instrument Software Architecture) is a multivendor I/O software standard approved by the IVI Foundation.[1] It provides a common foundation for the development, delivery, and interoperability of high-level multivendor system software components, such as instrument drivers, soft front panels, and application software.

The R&S VISA supports the Ethernet (VXI-11, HiSLIP, RSIB, Raw Socket), Serial (RS-232) and USB interfaces (USBTMC and R&S®NRP[2]) on Windows 7 (32/64-bit), Windows 8 (32/64-bit), Windows 10, OS X 10.10 or later, Linux Ubuntu 16.04 or later, and CentOS 7.

The objective of this document is to give a quick-start example (Chapter 3), introduce the utility applications, and to describe the C interface of the R&S VISA library (Chapter 8).

The R&S VISA contains three utility applications:

- The RsVisaTester allows to find resources and to communicate with devices. Most VISA functions can be invoked from this application (Chapter 4).
- With the RsVisaTraceTool all VISA function calls are logged (Chapter 5).
- To define aliases and resource list or to configure the conflict manager the RsVisaConfigure application is used (Chapter 6).

Related Documents are:

- VPP-4.3: The VISA Library
- VPP-4.3.5: VISA Shared Components
- VPP-4.3.6 VISA Implementation Specification for .NET
- 1MA208: Fast Remote Instrument Control with HiSLIP

[1] http://www.ivifoundation.org

[2] R&S®NRP support is only available for Windows and requires installation of the NRP Toolkit

2 Conventions Used in the Documentation

The following conventions are used throughout the R&S VISA User Manual:

Typographical conventions

Convention	Description
“Graphical user interface elements”	All names of graphical user interface elements both on the screen and on the front and rear panels, such as dialog boxes, softkeys, menus, options, buttons etc., are enclosed by quotation marks.
“KEYS”	Key names are written in capital letters and enclosed by quotation marks.
Input	Input to be entered by the user is displayed in italics.
`File names, commands, program code`	File names, commands, coding samples and screen output are distinguished by their font.
"Links"	Links that you can click are displayed in blue font.
"References"	References to other parts of the documentation are enclosed by quotation marks.

Other conventions

- **Remote commands**: Remote commands may include abbreviations to simplify input. In the description of such commands, all parts that have to be entered are written in capital letters. Additional text in lower-case characters is for information only.

3 Quick-Start Example

This chapter gives a small example how to use the VISA C library. The purpose of the example is to illustrate how to find and open resources, and how to send simple SCPI commands from a C++ application. The complete source code of this example can be found in the RsVisa section of the start menu and in the folder `%PUBLIC%\Documents\Rohde-Schwarz\RsVisa\Samples\C++\IdnSample`.

3.1 Creating a Visual Studio Solution

The example contains a Visual Studio 2013 solution. If you want to create a solution manually or use a different development environment set the include path such that the `visa.h` file is found; this is achieved by appending `%VXIPNPPATH%WinNT\RsVisa\include` to the include path. Likewise, the linker has to link against the `RsVisa32.lib` library which is available in `%VXIPNPPATH%WinNT\RsVisa\lib\msc` for 32-bit and `%VXIPNPPATH64%Win64\RsVisa\lib\msc` for 64-bit applications.

3.2 Creating an Xcode project for Mac

Since the example ships with a `CMakeLists.txt` file, you can use CMake to create an Xcode project. Open a console and follow these steps to create an Xcode project:

- Create a build folder
- Change into the builder folder
- Execute the command
  `cmake /Applications/Rohde-Schwarz/Example/C++/IdnSample`

Alternatively you may use the CMake GUI application.

Please refer to section 7.2 for more details on the CMake support of the R&S VISA.

3.3 Compiling the example on Linux

Since the example ships with a `CMakeLists.txt` file, you can use CMake to create makefiles. Open a console and follow these steps to create makefiles:

- Create a build folder
- Change into the builder folder
- Execute the command
  `cmake /usr/share/doc/rsvisa/Samples/IdnSample`

Alternatively you may use the CMake GUI application.

Please refer to section 7.2 for more details on the CMake support of the R&S VISA

3.4 Structure of the example

The example implements two classes:

- **VisaResourceManager** allows to find resources and to connect to devices by creating objects of the VisaSession class.
- **VisaSession** provides functions to write and read to an open session.

These classes wrap around the VISA C library calls. For example, creating an object of `VisaResourceManager` and invoking the member function `findResources` would be similar to the following code snippet:

```
ViSession rm;
viOpenDefaultRM(&rm);
std::vector <std::string> rsrcList;
ViUInt32 retCnt;
ViFindList vi;
ViChar desc[256];
ViAttrState searchAttributes = VI_RS_FIND_MODE_CONFIG |
VI_RS_FIND_MODE_VXI11 | VI_RS_FIND_MODE_MDNS;
viSetAttribute(rm, VI_RS_ATTR_TCPIP_FIND_RSRC_MODE, searchAttributes);
viFindRsrc(rm, "?*", &vi, &retCnt, desc);
rsrcList.push_back(desc);

for (ViInt16 i = 0; i < static_cast<ViInt16>(retCnt)-1; ++i) {
 viFindNext(vi, desc);
 rsrcList.push_back(desc); }

viClose(vi);
viClose(rm);
```

The basic steps in this code are:

- Creating a resource manager
- Setting the attributes in order to find network resources. Note that in order for this feature to be enabled the compiler macro `RSVISA_EXTENSION` has to be defined.
- Calling `viFindRsrc` to initialize the find list, get the number of found resources, and retrieve the description of the first result; subsequent calls of `viFindNext` retrieve all search results.
- Finally the handlers of the search list and of the resource manager are closed.

3.5 Using PyVISA with Linux

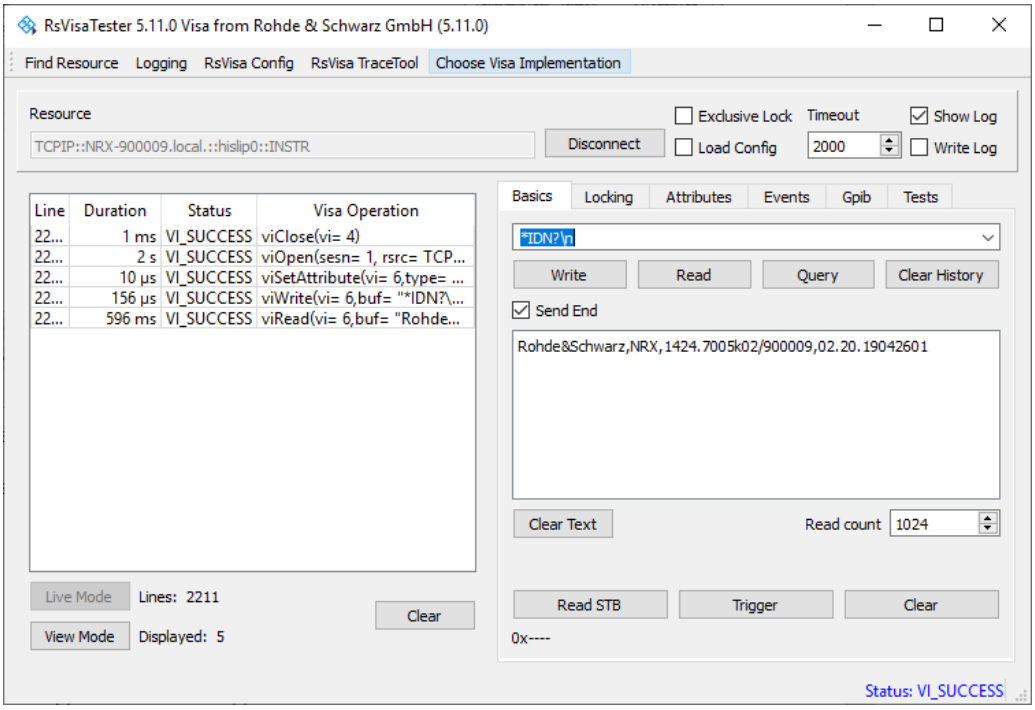
Please refer to the online manuals (e.g. https://pypi.org/project/PyVISA/) how you can install and use Python scripts with the VISA Library. The following sample script configures the PyVISA to use the R&S VISA Library directly in Linux:

```
#!/usr/bin/env python3
import visa

# Open VISA Resource-Manager
rm = visa.ResourceManager("/usr/lib/librsvisa.so@ni") #use this for Ubuntu
rm = visa.ResourceManager("/usr/lib64/librsvisa.so@ni") #use this for CentOS
#rm = visa.ResourceManager() #use this for Windows
print(rm)

# Show available resources
list = rm.list_resources()
print(list)

dev = rm.open_resource('TCPIP::example.com::hislip0')
#dev.write_termination = '\n' #use this for raw socket connections
#dev.read_termination = '\n'  #e.g. TCPIP::example.com::5025::SOCKET
dev.write("*IDN?")
idn = dev.read()
print("IDN:", idn)
dev.close()
```

4 RsVisaTester

The RsVisaTester application provides a simple way to call VISA functions from a PC application. Furthermore, it is also capable of running tests, which check the performance or reliability of a channel.

4.1 Main Window



The basic workflow of the RsVisaTester is to first find a resource, connecting, and then calling the desired VISA functions with their respective parameters. The following list gives an overview of the VISA functions called (for details of the functions refer to Sec. 9.1):

- “Connect”: `viOpen`
- “Write”: `viWrite`
- “Read”: `viRead`
- “Query”: `viWrite` and `viRead`
- “ReadSTB”: `viReadSTB`
- “Trigger”: `viAssertTrigger`
- “Clear”: `viClear`
- “Exclusive Lock” and “Shared Lock”: `viLock`
- “Unlock”: `viUnlock`
- “Go to Local” and “Go to Remote”: `viGpibControlREN`
- “Get Attribute”: `viGetAttribute`
- “Set Attribute”: `viSetAttribute`

- “Enabled Event”: `viEnabledEvent`
- “Discard Event”: `viDiscardEvents`
- “Disabled Event”: `viDisableEvents`
- “Wait On Event”: `viWaitOnEvent`
- “Install Handler”: `viInstallHandler`
- “Uninstall Handler”: `viUninstallHandler`
- “Send”: `viSendIFC`, `viGpibCommand` or `viGpibPassControl`

If the “Show Log” checkbox is checked an entry for each VISA function call appears in the log-view. If if the “Write Log” checkbox is checked the log-view entry is written to the log file as well. The log-view can be operated in two modes: the “Live Mode” shows only the most recent messages whereas the “View Mode” allows to scroll the history.

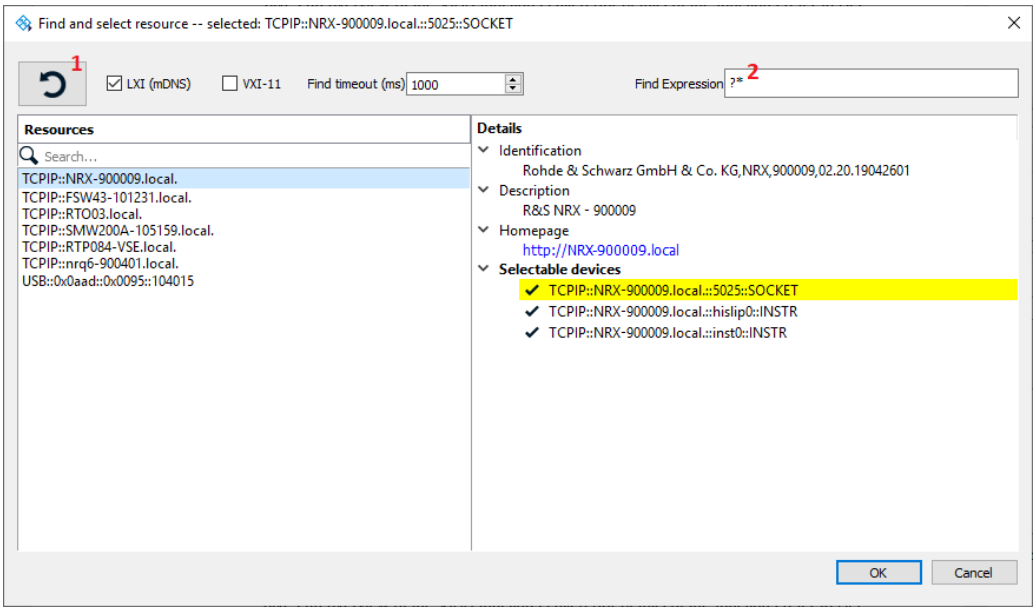
4.2 Tests

In the “Tests” panel of the main window one can start three different kind of tests, testing two different resources simultaneously:

- **Performance Tests**: Measures durations of some SCPI commands and data throughput.
- **Stress 4882 Test**: Tests reliability of channel by rapidly calling different VISA functions.
- **Stress Mmem Test**: Tests if channel is capable of handling large data transfers.

4.3 Find Resource Dialog

This dialog is displayed when clicking on “Find Resource” in the main window.

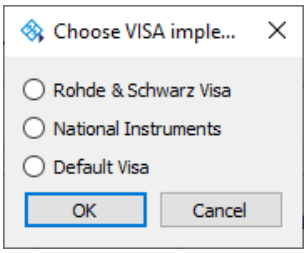


If you click **Refresh** button (1), the resource list is refreshed and the available resources are displayed grouped by devices. The LXI (mDNS) and VXI-11 search are only

available if the R&S VISA is loaded. The text field **Find Expression** (2) at the top contains the search expression as defined by the `viFindRsrc` function (cf. Sec. 9.1.2). Devices are grouped in the list on the left panel. Use the right **Details** panel for further information on LXI devices and select the device under “Selectable devices”. Use the search text field to quickly filter the current device list and details.

4.4 Choose VISA Implementation Dialog

This dialog can be accessed from the main window by clicking on “Choose Visa Implementation”.



All available VISA implementations are displayed. After selecting an implementation and pressing “OK” the current VISA library is unloaded and the selected VISA implementation is loaded. It is strongly recommended to make sure that there are no open connections when changing the VISA implementation.

If one chooses the “Default Visa” in the 64-bit Version of the RsVisaTester application the functionality of the VISA conflict manager is invoked (cf. Sec. 6.4).

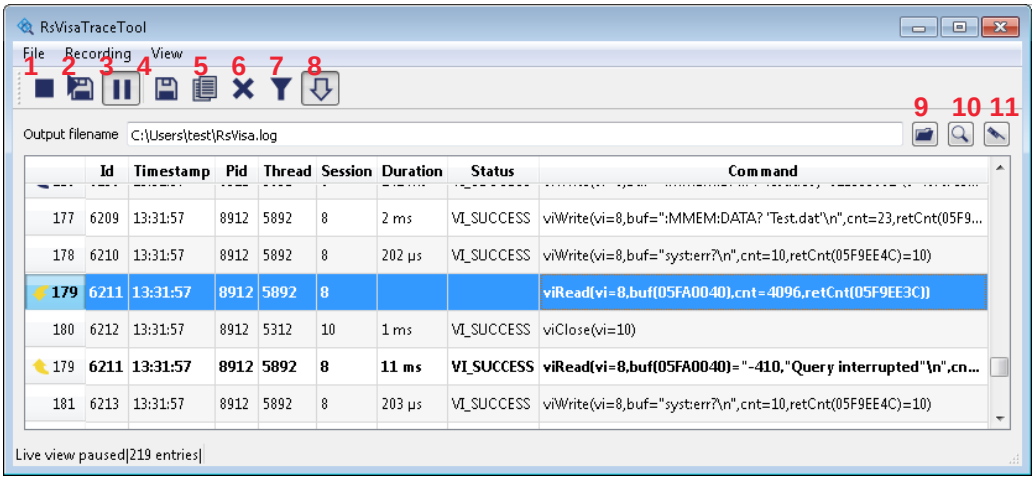
At startup the RsVisaTester loads the Rohde & Schwarz VISA implementation.

5 RsVisaTraceTool

The RsVisaTraceTool allows to log the communication between the VISA library and the PC application. It provides two means of tracing: i) recording to memory and ii) recording to a file. Recording to memory is a fast way to trace the communication allowing the definition of flexible filters. However, as the size of the memory is limited only the most recent VISA commands are kept. For long-time tracing the second mode, recording to a file, should be employed.

By starting the RsVisaTraceTool multiple times and setting up appropriate filters (Sec. 5.3) one can trace several PC applications independently.

5.1 Main Window



The main window shows a list of the captured commands and allows to control the recording. The labeled buttons provide the following functionality:

1. Toggle Start/Stop of recording to memory.
2. Toggle Start/Stop of recording to file.
3. Activates/Deactivates a pause. If a pause is active the recording continues but the list of captured commands is not refreshed. The menu item “View -> Pause on error” activates the pause if a VISA call returns with an error status. Deactivating the pause clears the list of captured commands.
4. Save the list of captured commands to a file.
5. Copy the list of captured commands to the windows clipboard.
6. Clear the list of captured commands.
7. Edit view filters (Sec. 5.3). If view filters are active the color of the icon is lightblue.
8. When capturing, scroll to the end of the list of captured commands.
9. Select log file.
10. Open destination of log file in windows explorer.
11. Delete log file.

Notice, that starting and stopping the recording to file or memory starts and stops the recording of all RsVisaTraceTool instances. However, pause and clear only apply to the current instance.

The list of captured commands contains the following fields:

- **Id:** Process independent identifier, which is the same in all RsVisaTraceTool instances
- **Timestamp**: Time of the system clock
- **Pid**: Process ID of the PC application
- **Thread**: ID of the thread making the VISA call
- **Session**: VISA session number
- **Address**: The VISA resource string used for this connection
- **Duration**: CPU time used by the VISA to execute command.
- **Status**: Return code of the VISA command.
- **Command**: VISA command called by the application with all parameters. If a parameter name is followed by a bracketed number, it is a pointer parameter and the number indicates the hex-coded memory address.

The first column of the table contains an ID, which is assigned by each process separately. Therefore, two different processes may assign the same IDs. Columns can be hidden or shown via a popup-menu accessible by right-clicking on the table header.

In the case of two or more threads (of the same or of different processes) running simultaneously it may occur that one thread invokes a VISA function while the other thread is still performing a VISA call, hence both calls are intervened. If between the start and the end of a VISA call of one thread the start or the end of a VISA call of a different thread occurs, two lines appear in the list of captured commands: one for the beginning of the VISA call and one for the ending.

For example, in the screenshot thread 5892 makes a `viRead` call which takes 11 ms. Within this timespan thread 5312 makes a `viClose` call. To indicate that the other thread makes an operation two lines for `viRead` appear: one (with no duration and status information) at the start of the operation and another one, after `viClose`, at the end.[3] The starting message shows an arrow-down and the ending message an arrow-up icon in the first column. If one selects a message with an arrow symbol the corresponding starting or ending message is displayed in a bold font type. Furthermore, double-clicking centers the corresponding line in the view.

If the menu entry “View -> Options -> Collapse same commands” is checked, lines containing the exact same commands from the same thread are displayed only once.

	Duration	Command
17	2 ms	10 x viReadSTB(vi=8,stb(04E2E148)=0x0000)

In this case a tree symbol is shown at the beginning of the line indicating that this line represents multiple commands. The durations of all collapsed lines are summed up and

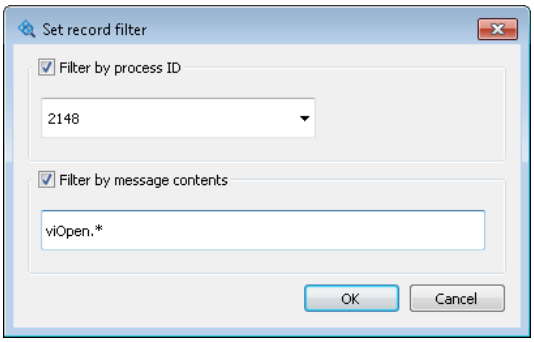
[3] For the `viClose` command only one line appears, because this command is not interrupted by another thread.

displayed. Furthermore, a counter preceding the actual command indicates how many commands were collapsed.

If the menu entry “View -> Options -> Show only string arguments” is checked, not all the arguments of a VISA call are displayed in the command column, but only string arguments.

5.2 Record Filter

The record filter dialog is opened by the “Recording -> Configure Filter” menu entry of the main window.



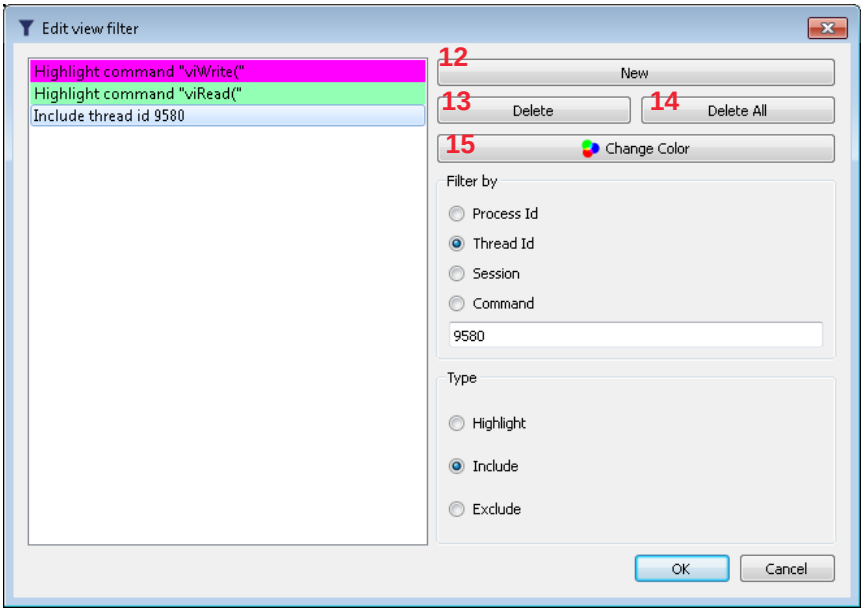
The filters defined by this dialog are processed in the VISA library. Hence, these filters are active for recording to memory of all RsVisaTraceTool instances and recording to file. As the filter is applied before the data is transmitted to the RsVisaTraceTool application one can save memory space by employing these filters.

If one defines a filter by process ID, only commands originating from the process with the given PID are recorded. If one defines a filter by message contents only commands matching the given string are recorded. The string is interpreted as a regular expression.[4]

5.3 Edit View Filter Dialog

The edit view filter dialog is opened by clicking “button 7”.

[4] For details of the regular expression syntax see http://www.cplusplus.com/reference/regex/ECMAScript/



The filters defined in this dialog apply to the current view. A convenient way to define filters is to right-click on the captured commands list of the main window and to use the popup menu to add a filter.

Each view filter either filters by process id, thread id, session number, VISA resource string, or a string contained in a command. As a result, it either highlights the commands or it includes or excludes the commands from the captured commands list. If the filter type is “exclude” no commands matching the filter are shown in the list. On the other hand, if there is at least one filter of the “include” type only commands matching an include-type filter are displayed. The filters are only applied to the current view; hence, one can change the filters without losing data.

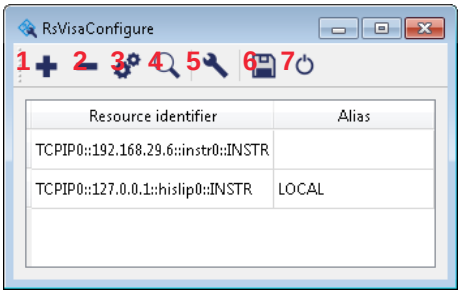
The functionality of the labeled buttons is the following:

12. Adds a filter.
13. Deletes the selected filter.
14. Deletes all filters.
15. Changes the color of the selected filter which is used to highlight matching commands.

6 RsVisaConfigure

The purpose of the RsVisaConfigure application is to define a list of resources - optionally with an alias - and to set GPIB properties. All resource identifiers are displayed when searching for VISA resources with `viFindRsrc`. Furthermore, if an alias is defined the alias can be used instead of the resource string e.g. when accessing the resource with `viOpen`.

6.1 Main Window



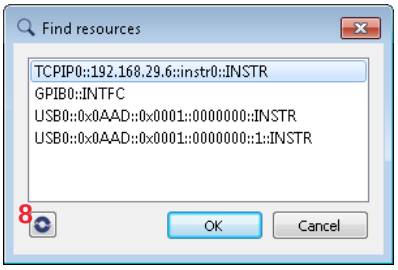
The screenshot shows the main window of the application. The labeled buttons provide the following functionality:

1. Add a new resource identifier to the list.
2. Remove the selected identifier from the list.
3. Edit the selected identifier (Sec. 6.3).
4. The selected resource identifier is replaced by an identifier chosen from a list of available resources (Sec. 0).
5. Edit GPIB and special properties (Sec. 6.4).
6. Save the current configuration.
7. Quit the application.

The resource identifier or alias can be edited by double clicking on the cell or by selecting the desired row and clicking “button 3”.

6.2 Find Resources Dialog

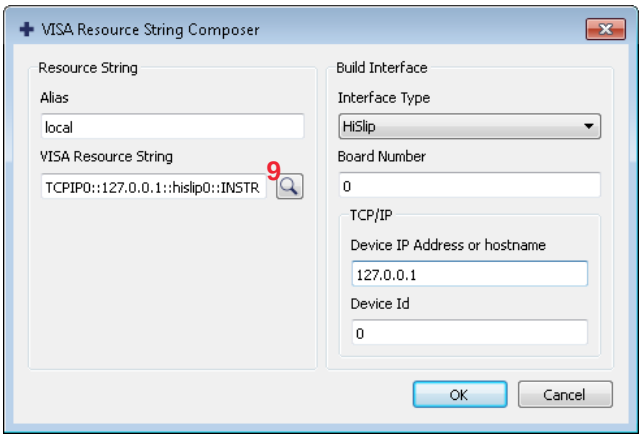
This dialog is used to select a VISA resource. It can be accessed by clicking on “button 4” in the main window or “button 9” in the resource string composer dialog.



By clicking on “button 8” the list of available resources is refreshed. Note, the search for available resources includes network devices which respond via VXI-11 or mDNS queries. Therefore, on large networks the search might take some time.

6.3 Resource String Composer

This dialog is used to edit the resource identifier and, optionally, to assign an alias. It is opened by clicking on “button 3” of the main window.

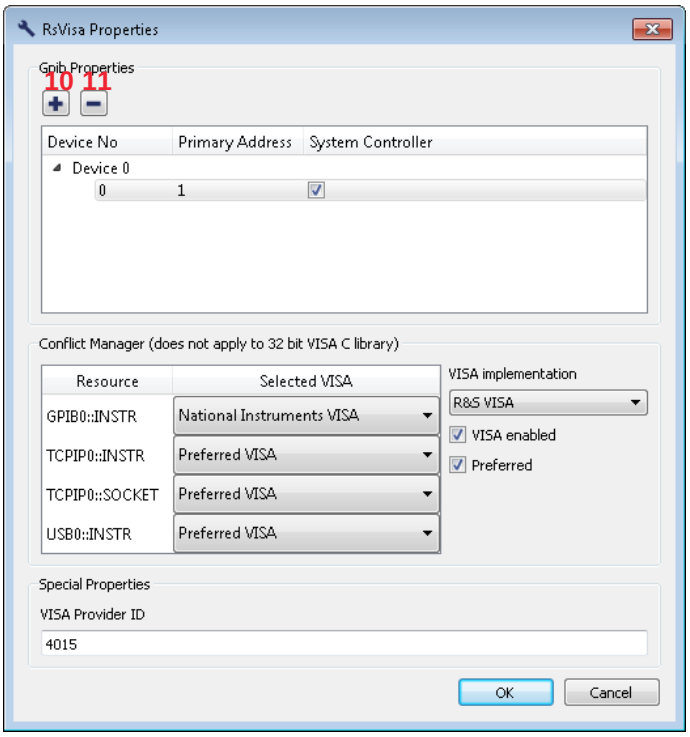


This dialog provides two ways to edit the VISA resource string: i) the user edits the resource string freely in the left panel or ii) the fields of the right panel are used to construct the resource string. When changing any fields of the right panel the resource string is updated overwriting the current string. Strings constructed by means of the right panel are guaranteed to be syntactically correct, but are not checked for validity. On the other hand, if the user edits the string freely no checks are employed.

By clicking on “button 9” the VISA resource string is replaced by an identifier chosen from a list of available resources (Sec. 0).

6.4 Property Dialog

This dialog is used to edit GPIB properties, change settings of the VISA conflict manager, and to set the VISA Provider ID. It is opened from the main window by clicking on “button 5”.



“Button 10” and “button 11” are used to add and remove devices, respectively. The properties “Device No” and “Primary Address” are edited by double clicking the desired cell; the property “System Controller” is set by checking the checkbox.

The VISA Conflict Manager provides means to switch between different VISA implementations and is part of the VISA shared components. Here the user can select a VISA implementation for each resource type. If a device of this resource is opened, the conflict manager loads the selected VISA implementation. Furthermore, a preferred VISA implementation can be defined, which is taken in the case that for a resource type no specific VISA implementation is selected. If a VISA implementation is disabled, the conflict manager does not load this VISA implementation. Changes in the Conflict Manager become effective after clicking “OK”. Note that the Conflict Manager is not invoked by 32-bit applications linked to the VISA C library.

For the example given in the screenshot, all connections to GPIB devices are handled by the National Instruments VISA; for connections to all other devices the preferred VISA is used, which is in this case the R&S VISA.

The VISA provider ID is by default the R&S VISA ID and should only be changed under special circumstances (e.g. if a third-party software requires a certain VISA provider ID). The value is returned by the attribute `VI_ATTR_RSRC_MANF_ID`. For a change of this property to become effective a restart of the applications using the VISA is required.

7 Developing with the R&S VISA

7.1 Switching between VISA implementations

7.1.1 Windows

The R&S VISA provides the proprietary `RsVisaLoader.dll` file, which is not part of the VISA standard. This library allows user applications to switch at runtime between VISA implementations of different vendors. Therefore the RsVisaLoader library forwards all VISA calls to the vendor specific VISA library. Hence, in addition to all exports of the `visa32.dll` library the `RsVisaLoader.dll` library exports functions to switch between implementations:

- `RsViSetDefaultLibrary`: Loads the VISA library of a specific vendor. Call this function before any other VISA function call.
- `RsViUnloadVisaLibrary`: Unloads the currently loaded VISA library.
- `RsViIsVisaLibraryInstalled`: Checks whether the implementation of a specific vendor is currently available.

For details of usage refer to the header file. The header and lib files, `RsVisaLoader.h` and `RsVisaLoader.lib`, are located in the directories:

- `%VXIPNPPATH%WinNT\RsVisa\lib\msc` (32-bit)
- `%VXIPNPPATH64%Win64\RsVisa\lib\msc` (64-bit)
- `%VXIPNPPATH%WinNT\RsVisa\include`

7.1.2 OS X

The R&S VISA provides a similar mechanism to switch between different VISA implementation at runtime for OS X as it does for Windows. The main difference is that, unlike for Windows, for OS X no compiled library is provided. The recommended way to use the RsVisaLoader features under Mac is to compile the sources directly in your project. The source files and an example are available at `/Applications/Rohde-Schwarz/RsVisaLoader`.

In case you need a library you can use the CMakeLists.txt file in that folder to create an XCode project which provides one library target. Compiling the XCode project produces the library against which you have to link your application. However, in this case you should consider deploying the RsVisaLoader dylib file with your application bundle.

The API for the loader mechanism is defined in the header file `/Applications/Rohde-Schwarz/RsVisaLoader/RsVisaLoaderMac.h`. The function calls to switch between VISA implementations are identical to the Windows API.

7.2 CMake support

The R&S VISA deploys CMake[5] configuration files which let you easily include the R&S VISA library in your CMake project. An example for this technique is given in the C++ example which is deployed with the R&S VISA.

You can make a VISA library available in your CMake project by adding the line

```
find_package(RsVisa REQUIRED)
```

to your CMakeLists.txt file. This command defines the following targets:

- **rsvisa::rsvisa** Imports the R&S VISA library
- **rsvisa::loader** Imports the R&S VISA loader (only on Windows)
- **rsvisa::visa** Imports the standard VISA (visa32.lib/ visa64.lib, only on Windows)

For example, if you want your application to link against the R&S VISA library you have to add the following line (or similar) to your CMakeLists.txt file:

```
target_link_libraries(${PROJECT_NAME} rsvisa::rsvisa)
```

Since the target's properties already contain the include path to the VISA headers you do not need to set include directories for the VISA.

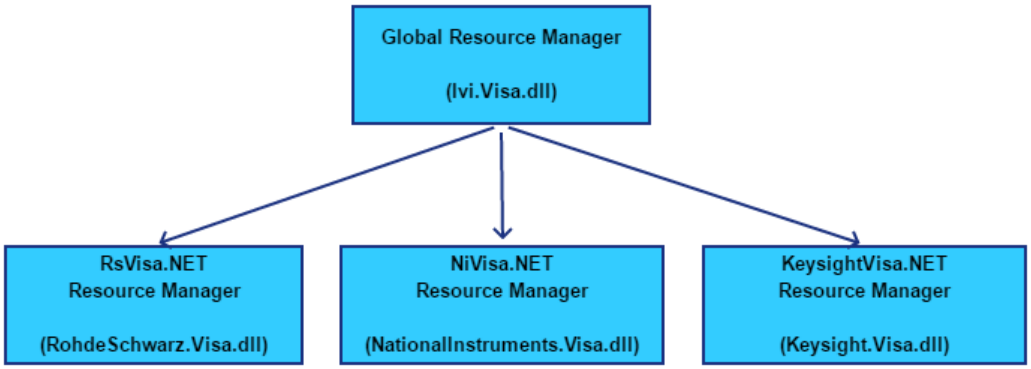
It is only recommended to link against the target `rsvisa::rsvisa` if you need R&S VISA specific features like device discovery over LAN. If you want to stay flexible and switch between VISA implementations during runtime you should use the target `rsvisa::loader`. However, in most cases applications do not depend on VISA specifics and in this case it is advisable to link against the default VISA by using the target `rsvisa::visa`.

[5] cmake.org/

8 VISA.NET

R&S VISA.NET is a part of our Windows Developer Installation packet. It offers native C# interface according the IVI VISA.NET specification (http://www.ivifoundation.org/specifications/default.aspx VPP-4.3.6).

You can use R&S VISA.NET in two different ways – through Global Resource Manager (GRM), or directly. The picture below shows the relationship between the R&S VISA.NET Resource Manager (RM) and the GRM:



As a part of the R&S VISA.NET installation we provide four C# examples, two for each approach. You can find them here:

```
Windows Start Menu -> R&S VISA -> Samples -> C#
```

The examples whose names start with **`VisaDotNet`** use the GRM.

The examples starting with **`RsVisaDotNet`** with use the RsVisa.NET RM directly.

8.1 Using of the IVI VISA.NET Global Resource Manager

We recommend this approach, since it is the most universal way supported by all VISA.NET vendors.

Here, you do not call any vendor-specific VISA.NET implementation. Instead, you call the GRM with a request to open a session to your resource (instrument).

The GRM is a part of IVI VISA.NET shared components installed together with the R&S VISA.NET. The assembly is built as **Any CPU**. In 64-bit Windows it is copied to both 32-bit and 64-bit paths. You have to add the Ivi.Visa.dll to your project.

File location (64-bit Windows):

```
c:\Program Files\IVI Foundation\VISA\Microsoft.NET\Framework64\
v2.0.50727\VISA.NET Shared Components x.x.x\Ivi.Visa.dll
```

File location (32-bit Windows):

```
c:\Program Files (x86)\IVI Foundation\VISA\Microsoft.NET\Framework64\
v2.0.50727\VISA.NET Shared Components x.x.x\Ivi.Visa.dll
```

The GRM asks each vendor-specific Resource Manager (RM) if they support the requested resource. The first RM that answers positively, gets the control over the session.

Examples using the GRM approach: `Start Menu -> R&S VISA -> Samples -> C#`

- `VisaDotNet_IdnQuery` – console application sending *IDN? query
- `VisaDotNet_NrpzMeasure` – console application that performs a measurement with R&S NRP-Zxx power sensors

8.2 Direct use of the R&S VISA.NET Resource Manager

In some cases, you want to use the vendor-specific implementation directly. The most common reasons are:

- You want to have more control over the different software components used in your application
- You want to use a specialty of an implementation that goes beyond the IVI VISA.NET specification

In our two examples we show the specialty of the R&S VISA.NET resource manager to be able to find resources via VXI-11 broadcast and mDNS/Bonjour (`Start Menu -> R&S VISA -> Samples -> C#`):

- `RsVisaDotNet_FindLxi` – simple console application searching for all the instruments discoverable by VXI-11 and mDNS
- `RsVisaDotNet_FindLxiWithGui` – Windows Forms application searching for all the discoverable instruments. You can switch VISA implementations between the R&S VISA.NET and NI VISA.NET (if installed).

Location of the R&S VISA.NET assembly:

R&S VISA.NET assembly is built as **Any CPU**. In 64-bit Windows the assembly is copied to both 32-bit and 64-bit paths.

File location (64-bit Windows):

```
c:\Program Files (x86)\IVI Foundation\VISA\Microsoft.NET\Framework32\
v4.0.30319\RS VISA.NET x.x.x\RohdeSchwarz.Visa.dll
```

File location (32-bit Windows):

```
c:\Program Files (x86)\IVI Foundation\VISA\Microsoft.NET\Framework32\
v4.0.30319\RS VISA.NET x.x.x\RohdeSchwarz.Visa.dll
```

9 VISA C Library

9.1 String Formats

This section describes the required formats of VISA address strings, search expressions, and format strings.

9.1.1 VISA Address Strings

The following table shows the grammar for the Address String. Optional string segments are shown in square brackets (“[]”).

Interface	Grammar
ASRL	ASRL[*board*][::INSTR]
TCPIP	TCPIP[*board*]::*host_address*[::*LAN_device_name*][::INSTR]
TCPIP	TCPIP[board]::*host_address*[::*HiSLIP_device_name*[,*HiSLIP_port*]][::INSTR]
TCPIP	TCPIP[*board*]::*host_address*::*port*::SOCKET
USB	USB[*board*]::*manufacturer_ID*::*model_code*::*serial_number*[::*USB_interface_number*][::INSTR]
RSNRP	RSNRP::*model_code*::*serial_number*

The ASRL keyword is used to establish communication with an asynchronous serial (such as RS-232) device. The TCPIP keyword is used to establish communication with Ethernet instruments. The USB keyword is used to establish communication with USB instruments.

The default values for optional string segments are shown below.

Optional String	Default Value
board	0
LAN_device_name	inst0
HiSLIP_device_name	hislip0
HiSLIP_port	4880
USB_interface_number	lowest numbered relevant interface

The expressions “**address string**”, “**resource string**”, “**resource identifier**”, or in this context sometimes just “**resource**” are used synonymously in this document.

Examples for address string:

- `ASRL1::INSTR` - serial device located on port 1
- `TCPIP0::1.2.3.4::5025::SOCKET` – Raw TCP/IP access to port 5025 at the specified address.
- `TCPIP::devicename.company.com::INSTR` – TCP/IP device using VXI-11 located at the specified address. This uses the default LAN Device Name of `inst0`.
- `TCPIP::1.2.3.4::inst0::INSTR` – A TCP/IP device using VXI-11 located at the IP address 1.2.3.4.
- `TCPIP::127.0.0.1::hislip0::INSTR` – TCP/IP device using HiSLIP located at IP address 127.0.01.
- `USB::0x1234::0x5678::A22-5::INSTR` – USB Test & Measurement class device with manufacturer ID 0x1234, model code 0x5678, and serial number A22-5. This uses the device’s first available USBTMC interface, usually number 0.
- `USB::0x0AAD::0x0095::104015::INSTR` – R&S NRP-Zxx legacy Powersensor model 0x0095 (NRP-Z86), serial number 104015
- `RSNRP::0x0095::104015::INSTR` – alias to the abovementioned `USB::0x0AAD::0x0095::104015::INSTR`

Ipv6 is only supported in HiSLIP and SOCKET sessions

Examples for IPv6 address string:

- `TCPIP::[::1]::hislip0::INSTR` – TCP/IP device with IPv6 using HiSLIP on localhost
- `TCPIP::[fe80::ad82:1033:398b:c921]::hislip0::INSTR` - TCP/IP device with IPv6 using HiSLIP
- `TCPIP0::[fe80::ad82:1033:398b:c921]::5025::SOCKET` – Raw TCP/IP access with IPv6 to port 5025 at the specified address.

9.1.2 viFindRsrc Expressions

The syntax of the `expr` parameter of the `viFindRsrc` command (cf. Sec. 9.2.10) is a regular expression, which is a string consisting of ordinary characters as well as special characters. A regular expression is used for specifying patterns to match in a given string. Given a string and a regular expression, one can determine if the string matches the regular expression. A regular expression can also be used as a search criterion. Given a regular expression and a list of strings, one can match the regular expression against each string and return a list of strings that match the regular expression.

The following two tables define the special characters and literals used in the grammar rule:

Character	Description	Symbol
NL / LF	New Line / Line Feed	"\n"
HT	Horizontal Tab	"\t"
CR	Carriage Return	"\r"
FF	Form Feed	"\f"
SP	Blank Space	" "

Literal	Definition
white_space	NL, LF, HT, CR, FF, SP
digit	"0","1".."9"
letter	"a","b"..."z", "A","B"..."Z"
hex_digit	"0","1"..."9", "a","b"..."f", "A","B"..."F"
underscore	"_"

For regular expression special characters and operators are used as follows:

Special Characters and Operators	Meaning
?	Matches any one character.
\	Makes the character that follows it an ordinary character instead of special character. For example, when a question mark follows a backslash (i.e. '\?'), it matches the '?' character instead of any one character.
[*list*]	Matches any one character from the enclosed *list*. A hyphen can be used to match a range of characters.
[^*list*]	Matches any character not in the enclosed *list*. A hyphen can be used to match a range of characters.
*	Matches 0 or more occurrences of the preceding character or expression.
+	Matches 1 or more occurrences of the preceding character or expression.
exp|*exp*	Matches either the preceding or following expression. The OR operator “|” matches the entire expression that precedes or follows it and not just the character that precedes or follows it. For example, “USB|TCPIP” means “(USB)|(TCPIP)”, not “US(B|T)CPIP”.
(*exp*)	Grouping characters or expressions.

Some examples are:

Regular Expression	Sample Matches
`?*`	Matches all resources.
`TCPIP?*INSTR`	Matches `TCPIP0::127.0.0.1::inst0::INSTR`, `TCPIP1::192.168.0.1::hislip0::INSTR` but not `TCPIP0::1.2.3.4::999::SOCKET`
`ASRL[0-9]*::?*INSTR`	Matches `ASRL1::INSTR` but not `TCPIP::127.0.0.1::INSTR`.
`(TCPIP|USB)?*INSTR`	Matches LAN (VXI-11 & HiSLIP) and USBTMC instruments but not raw socket or serial devices.
`?*INSTR`	Matches all `INSTR` (device) resources.
`RSNRP?*`	Matches all the R&S NRP-Zxx legacy

9.1.3 Format String for viPrintf functions

The format strings, as presented in this section, are employed in the `viPrintf` (cf. Sec. 9.2.24) and their derivatives (`viQueryf`, `viSPrintf`, `viVPrintf`, `viVSPrintf`, and `viVQueryf`).

In these commands the parameter `writeFmt` (or equivalent) string can include regular character sequences, special formatting characters, and special format specifiers. The regular characters (including white spaces) are written to the device unchanged. The special characters consist of “\” (backslash) followed by a character. The format specifier sequence consists of “%” (percent) followed by an optional modifier (flag), followed by a format code.

9.1.3.1 Special Formatting Characters

Special formatting character sequences send special characters. The following table lists the special characters and describes what they send to the device.

Formatting Character	Character Sent to Device
\n	Sends the ASCII LF character. The END identifier will also be automatically sent.
\r	Sends an ASCII CR character.
\t	Sends an ASCII TAB character.
\###	Sends the ASCII character specified by the octal value.
\"	Sends the ASCII double-quote (") character.

\\	Sends a backslash (“\”) character.

9.1.3.2 Format Specifiers

The format specifiers convert the next parameter in the sequence according to the modifier and format code, after which, the formatted data is written to the specified device. The format specifier takes the following syntax:

%[modifiers]*format code*

where *format code* specifies the data type in which the argument is represented. Modifiers are optional codes that describe the target data.

In the following tables, a “d” format code refers to all conversion codes of type *integer* (“d”, “l”, “o”, “u”, “x”, and “X”), unless specified as %d only. Similarly, an “f” format code refers to all conversion codes of type *float* (“f”, “e”, “E”, “g”, “G”), unless specified as %f only.

Every conversion command starts with the “%” character and ends with a conversion character (format code). Between the “%” character and the format code, the following modifiers can appear in the sequence:

Modifier	**Supported with Format Code**	**Description**
An integer specifying *field width*.	d, f, s format codes	This specifies the minimum field width of the converted argument. If an argument is shorter than the *field width*, it will be padded on the left (or on the right if the - flag is present). Special case: For the “@H”, “@Q”, and “@B” flags, the *field width* includes the “#H”, “#!”, and “#B” strings, respectively. A “*” may be present in lieu of a field width modifier, in which case an extra `arg` is used. This `arg` must be an integer representing the *field width*.
An integer specifying *precision*.	d, f, s format codes	The *precision* string consists of a string of decimal digits. A “.” (decimal point) must prefix the *precision* string. The *precision* string specifies the following: • The minimum number of digits to appear for the “@1”, “@H”, “@Q”, and “@B” flags and the “i”, “o”, “u”, “x”, and “X” format codes. • The maximum number of digits after the decimal point in case of “f” format codes. • The maximum numbers of characters for the string (s) specifier. • Maximum significant digits for g format code. An asterisk “*” may be present in lieu of a *precision* modifier, in which case an extra `arg` is used. This `arg` must be an integer representing the *precision* of a numeric field.
An argument length modifier.	h (d, b, B format codes)	The argument length modifiers specify one of the following: • The “h” modifier promotes the argument to a short or unsigned short, depending on the format code type.

h, l, ll, L, z, and Z are legal values. (z and Z are not ANSI C standard flags.)	l (d, f, b, B format codes) L (f format code) z, Z (b, B format codes)	• The “l” modifier promotes the argument to a long or unsigned long. • The “ll” modifier promotes the argument to a long long or unsigned long long. • The “L” modifier promotes the argument to a long double parameter. • The “z” modifier promotes the argument to an array of floats. • The “Z” modifier promotes the argument to an array of doubles.
A comma (“,”) followed by an integer *n*, where *n* represents the array size.	%d (plus variants) and %f only	The corresponding argument is interpreted as a reference to the first element of an array of size *n*. The first *n* elements of this list are printed in the format specified by the format code. An asterisk (“*”) may be present after the “,” modifier, in which case an extra `arg` is used. This `arg` must be an integer representing the array size of the given type.
@1	%d (plus variants) and %f only	Converts to an IEEE 488.2 defined NR1 compatible number, which is an integer without any decimal point (for example, 123).
@2	%d (plus variants) and %f only	Converts to an IEEE 488.2 defined NR2 compatible number. The NR2 number has at least one digit after the decimal point (for example, 123.45).
@3	%d (plus variants) and %f only	Converts to an IEEE 488.2 defined NR3 compatible number. An NR3 number is a floating point number represented in an exponential form (for example, 1.2345E-67).
@H	%d (plus variants) and %f only	Converts to an IEEE 488.2 defined <HEXADECIMAL NUMERIC RESPONSE DATA>. The number is represented in a base of 16 form. Only capital letters should represent numbers. The number is of form "#H*XXX*.." where *XXX*.. is a hexadecimal number (for example, #HAF35B).
@Q	%d (plus variants) and %f only	Converts to an IEEE 488.2 defined <OCTAL NUMERIC RESPONSE DATA>. The number is represented in a base of eight form. The number is of the form "#Q*YYY*.." where *YYY*.. is an octal number (for example, #Q71234).
@B	%d (plus variants) and %f only	Converts to an IEEE 488.2 defined <BINARY NUMERIC RESPONSE DATA>. The number is represented in a base two form. The number is of the form "#B*ZZZ*.." where *ZZZ*.. is a binary number (for example, #B011101001).

9.1.3.3 Standard ANSI C Format Codes

- **%:** Send the ASCII percent (%) character.
- **c:** Argument type: A character to be sent.
- **d:** Argument type: An integer.

Modifier	**Interpretation**
Default functionality	Print an integer in NR1 format (an integer without a decimal point).

@2 or @3	The integer is converted into a floating point number and output in the correct format.
field width	Minimum field width of the output number. Any of the six IEEE 488.2 modifiers can also be specified with *field width*.
Length modifier l	`arg` is a long integer.
Length modifier ll	`arg` is a long long integer
Length modifier h	`arg` is a short integer.
, array size	`arg` points to an array of integers (or long or short integers, depending on the length modifier) of size array size. The elements of this array are separated by array size - 1 commas and output in the specified format.

- **f** Argument type: A floating point number.

Modifier	**Interpretation**
Default functionality	Print a floating point number in NR2 format (a number with at least one digit after the decimal point).
@1	Print an integer in NR1 format. The number is truncated.
@3	Print a floating point number in NR3 format (scientific notation). *Precision* can also be specified.
field width	Minimum field width of the output number. Any of the six IEEE 488.2 modifiers can also be specified with *field width*.
Length modifier l	`arg` is a double float.
Length modifier L	`arg` is a long double.
, array size	`arg` points to an array of floats (or doubles or long doubles), depending on the length modifier) of size array size. The elements of this array are separated by array size – 1 commas and output in the specified format.

- **s** Argument type: A reference to a NULL-terminated string that is sent to the device without change.

9.1.3.4 Enhanced Format Codes

- **b** Argument type: A location of a block of data.

Flag or Modifier	**Interpretation**
Default functionality	The data block is sent as an IEEE 488.2 <DEFINITE LENGTH ARBITRARY BLOCK RESPONSE DATA>. A count (long integer) must appear as a flag that specifies the number of elements (by default, bytes) in the block. A *field width* or *precision* modifier is not allowed with this format code.
* (asterisk)	An asterisk may be present instead of the count. In such a case, two `args` are used, the first of which is a long integer specifying the count of the number of

	elements in the data block. The second `arg` is a reference to the data block. The size of an element is determined by the optional length modifier (see below), default being byte width.
Length modifier h	The data block is assumed to be an array of unsigned short integers (16 bits). The count corresponds to the number of words rather than bytes. The data is swapped and padded into standard IEEE 488.2 format, if native computer representation is different.
Length modifier l	The data block is assumed to be an array of unsigned long integers. The count corresponds to the number of longwords (32 bits). Each longword data is swapped and padded into standard IEEE 488.2 format, if native computer representation is different.
Length modifier ll	The data block is assumed to be an array of unsigned long long integers. The count corresponds to the number of longlongwords (64 bits). Each longlongword data is swapped and padded into standard IEEE 488.2 format, if native computer representation is different.
Length modifier z	The data block is assumed to be an array of floats. The count corresponds to the number of floating point numbers (32 bits). The numbers are represented in IEEE 754 format, if native computer representation is different.
Length modifier Z	The data block is assumed to be an array of doubles. The count corresponds to the number of double floats (64 bits). The numbers will be represented in IEEE 754 format, if native computer representation is different.

- **B** Argument type: A location of a block of data. The functionality is similar to **b,** except the data block is sent as an IEEE 488.2 <INDEFINITE LENGTH ARBITRARY BLOCK RESPONSE DATA>. This format involves sending an ASCII LF character with the END indicator set after the last byte of the block.
- **y** Argument type: A location of a block of binary data.

Flags or Modifiers	**Interpretation**
Default functionality	The data block is sent as raw binary data. A count (long integer) must appear as a flag that specifies the number of elements (by default, bytes) in the block. A *field width* or *precision* modifier is not allowed with this format code.
* (asterisk)	An asterisk may be present instead of the count. In such a case, two `args` are used, the first of which is a long integer specifying the count of the number of elements in the data block. The second `arg` is a reference to the data block. The size of an element is determined by the optional length modifier (see below), default being byte width.
Length modifier h	The data block is an array of unsigned short integers (16 bits). The count corresponds to the number of words rather than bytes. If the optional “!ol” byte order modifier is present, the data is sent in little endian format; otherwise, the data is sent in standard IEEE 488.2 format. Data will be byte swapped and padded as appropriate if native computer representation is different.
Length modifier l	The data block is an array of unsigned long integers (32 bits). The count corresponds to the number of longwords rather than bytes. If the optional “!ol” byte order modifier is present, the data is sent in little endian format; otherwise, the data is sent in standard IEEE 488.2 format. Data will be byte swapped and padded as appropriate if native computer representation is different.

Length modifier ll	The data block is an array of unsigned long long integers (64 bits). The count corresponds to the number of longlongwords rather than bytes. If the optional “!ol” byte order modifier is present, the data is sent in little endian format; otherwise, the data is sent in standard IEEE 488.2 format. Data will be byte swapped and padded as appropriate if native computer representation is different.
Byte order modifier !ob	Data is sent in standard IEEE 488.2 (big endian) format. This is the default behavior if neither “!ob” nor “!ol” is present.
Byte order modifier !ol	Data is sent in little endian format.

The END indicator is not appended when LF(\n) is part of a binary data block, as with %b or %B.

9.1.3.5 BNF Format for viPrintf()

The following is the Backus-Naur-Form (BNF) format for the `viPrintf() writeFmt` string:

<print_fmt> := {<slashed_special> | <conversion> | <ascii_char> }*

<slashed_special> := "\n" | "\r" | "\\" | "\t" | <oct_esc> | "\"

<oct_esc> := "\"<oct_digit> [<oct_digit> [<oct_digit>]]

<ascii_char> := ASCII characters (other than backslash (\), percent (%), and NULL).

<conversion> := <fmt_cod_d> | <fmt_cod_f> | <fmt_cod_c> | <fmt_cod_b> | <fmt_cod_B> | <fmt_cod_s> | <fmt_cod_e> | <fmt_cod_y> "%%"

<fmt_cod_d> := "%" [<numeric_mod>] [<field width>] ["." <precision>] ["," <array_size>] ["l" | “ll” | "h"] "d"

<fmt_cod_f> := "%" [<numeric_mod>] [<field_width>] ["." <precision>] ["," <array_size>] ["l" |"L"] "f"

<fmt_cod_e> := "%" [<numeric_mod>] [<field_width>] ["." <precision>] ["," <array_size>] ["l" |"L"] "e"

<fmt_cod_b> := "%" <array_size> ["h" | "l" | “ll” | "z" |"Z"] "b"

<fmt_cod_B> := "%" <array_size> ["h" | "l" | “ll” | "z" | "Z"] "B"

<fmt_cod_c> := "%c"

<fmt_cod_s> := "%" [<just_mod>] [<field_width>] ["."<precision>] "s"

<fmt_cod_y> := "%" <array_size> [<swap_mod>] ["h" | "l" | “ll”] "y"

<swap_mod> := "!ob" | "!ol"

<numeric_mod> := "-" | "+" | " " | "@1" | "@2" | "@3" | "@H" | "@Q" | "@B"

<just_mod> := "-"

<field_width> := <positive_integer> | "*"

<precision> := <positive_integer> | "*"

<array_size> := <positive_integer> | "*"

9.1.4 Format String for viScanf functions

The format strings, as presented in this section, are employed in `viScanf` (cf. Sec. 9.2.29) commands and their derivatives (`viQueryf`, `viSScanf`, `viVScanf`, `viVSScanf`, and `viVQueryf`).

In these commands the parameter `readFmt` (or equivalent) string can include regular character sequences, special formatting characters, and special format specifiers. The white characters - blank, vertical tabs, horizontal tabs, form feeds, new line/linefeed, and carriage return - are ignored except in the case of `%c` and `%[ ]`. All other ordinary characters except `%` should match the next character read from the device. The format specifier sequence consists of "%" (percent) followed by optional modifier flags, followed by a format code.

9.1.4.1 ANSI C Standard Modifiers

Modifier	Supported with Format Codes	Description
An integer representing the *field width*	%s, %c, %[ ] format codes	It specifies the maximum field width that the argument will take. A '#' may also appear instead of the integer *field width*, in which case the next `arg` is a reference to the *field width*. This `arg` is a reference to an integer for %c and %s. The *field width* is not allowed for %d or %f.
A length modifier ('l,' 'll,' 'h,' 'z,' or 'Z'). z and Z are not ANSI C standard modifiers.	h (d, b format codes) l (d, f, b format codes) ll (d, b format codes) L (f format code) z, Z (b format code)	The argument length modifiers specify one of the following: a. The h modifier promotes the argument to be a reference to a short integer or unsigned short integer, depending on the format code. b. The l modifier promotes the argument to point to a long integer or unsigned long integer. c. The ll modifier promotes the argument to point to a long long integer or unsigned long long integer. d. The L modifier promotes the argument to point to a long double floats parameter. e. The z modifier promotes the argument to point to an array of floats. f. The Z modifier promotes the argument to point to an array of double floats.
* (asterisk)	All format codes	An asterisk acts as the assignment suppression character. The input is not assigned to any parameters and is discarded.

9.1.4.2 Enhanced Modifiers to ANSI C Standards

Modifier	Supported with Format Codes	Description
A comma (‘,’) followed by an integer *n*, where *n* represents the array size.	%d (plus variants) and %f only	The corresponding argument is interpreted as a reference to the first element of an array of size *n*. The first *n* elements of this list are printed in the format specified by the format code. A number sign (‘#’) may be present after the ‘,’ modifier, in which case an extra `arg` is used. This `arg` must be an integer representing the array size of the given type.
@1	%d (plus variants) and %f only	Converts to an IEEE 488.2 defined NR1 compatible number, which is an integer without any decimal point (for example, 123).
@2	%d (plus variants) and %f only	Converts to an IEEE 488.2 defined NR2 compatible number. The NR2 number has at least one digit after the decimal point (for example, 123.45).
@H	%d (plus variants) and %f only	Converts to an IEEE 488.2 defined <HEXADECIMAL NUMERIC RESPONSE DATA>. The number is represented in a base of sixteen form. Only capital letters should represent numbers. The number is of form "#H*XXX*.." where *XXX*.. is a hexadecimal number (for example, #HAF35B).
@Q	%d (plus variants) and %f only	Converts to an IEEE 488.2 defined <OCTAL NUMERIC RESPONSE DATA>. The number is represented in a base of eight form. The number is of the form "#Q*YYY*.." where *YYY*.. is an octal number (for example, #Q71234).
@B	%d (plus variants) and %f only	Converts to an IEEE 488.2 defined <BINARY NUMERIC RESPONSE DATA>. The number is represented in a base two form. The number is of the form "#B*ZZZ*.." where *ZZZ*.. is a binary number (for example, #B011101001).

9.1.4.3 Standard ANSI C Format Codes

- **c** Argument type: A reference to a character.

Flags or Modifiers	Interpretation
Default functionality	A character is read from the device and stored in the parameter.
field width	*field width* number of characters are read and stored at the reference location (the default *field width* is 1). No NULL character is added at the end of the data block.

Note: White space in the device input stream is *not* ignored.

- **d** Argument type: A reference to an integer.

Flags or Modifiers	Interpretation
Default functionality	Characters are read from the device until an entire number is read. The number read may be in either IEEE 488.2 formats <DECIMAL NUMERIC PROGRAM DATA>, also known as NRf; flexible numeric representation (NR1, NR2, NR3...); or <NON-DECIMAL NUMERIC PROGRAM DATA> (#H, #Q, and #B).
field width	The input number will be stored in a field at least this wide.
Length modifier l	`arg` is a reference to a long integer.
Length modifier ll	`arg` is a reference to a long long integer.
Length modifier h	`arg` is a reference to a short integer. Rounding is performed according to IEEE 488.2 rules (0.5 and up).
, array size	`arg` points to an array of integers (or long or short integers, depending on the length modifier) of size array size. The elements of this array should be separated by commas. Elements will be read until either array size number of elements are consumed or they are no longer separated by commas.

- **f** Argument type: A reference to a floating point number.

Flags or Modifiers	Interpretation
Default functionality	Characters are read from the device until an entire number is read. The number read may be in either IEEE 488.2 formats <DECIMAL NUMERIC PROGRAM DATA> (NRf) or <NON-DECIMAL NUMERIC PROGRAM DATA> (#H, #Q, and #B).
field width	The input number will be stored in a field at least this wide.
Length modifier l	`arg` is a reference to a double floating point number.
Length modifier L	`arg` is a reference to a long double number.
, array size	`arg` points to an array of floats (or double or long double, depending on the length modifier) of size array size. The elements of this array should be separated by commas. Elements will be read until either array size number of elements are consumed or they are no longer separated by commas.

- **s** Argument type: A reference to a string.

Flags or Modifiers	Interpretation
Default functionality	All leading white space characters are ignored. Characters are read from the device into the string until a white space character is read.
field width	This flag gives the maximum string size. If the *field width* contains a # sign, two arguments are used. The first argument read is a pointer to an integer specifying the maximum array size. The second should be a reference to an array. In case of *field width* characters already read before encountering a white space, additional characters are read and discarded until a white space character is found. In case of # *field width*, the actual number of characters that were copied into the user array, not counting the trailing NULL character, are stored back in the integer pointed to by the first argument.

9.1.4.4 Enhanced Format Codes

- **b** Argument type: A reference to a data array.

Flags or Modifiers	Interpretation
Default functionality	The data must be in IEEE 488.2 <ARBITRARY BLOCK PROGRAM DATA> format. The format specifier sequence should have a flag describing the *array size*, which will give a maximum count of the number of bytes (or words or longwords, depending on length modifiers) to be read from the device. If the *array size* contains a # sign, two arguments are used. The first argument read is a pointer to a long integer specifying the maximum number of elements that the array can hold. The second one should be a reference to an array. Also, in this case the actual number of elements read is stored back in the first argument. In absence of length modifiers, the data is assumed to be of byte-size elements. In some cases, data might be read until an END indicator is read.
Length modifier h	The array is assumed to be an array of 16-bit words, and count refers to the number of words. The data read from the interface is assumed to be in IEEE 488.2 byte ordering. It will be byte swapped and padded as appropriate to native computer format.
Length modifier l	The array is assumed to be a block of 32-bit longwords rather than bytes, and count now refers to the number of longwords. The data read from the interface is assumed to be in IEEE 488.2 byte ordering. It will be byte swapped and padded as appropriate to native computer format.
Length modifier ll	The array is assumed to be a block of 64-bit longlongwords rather than bytes, and count now refers to the number of longlongwords. The data read from the interface is assumed to be in IEEE 488.2 byte ordering. It will be byte swapped and padded as appropriate to native computer format.
Length modifier z	The data block is assumed to be a reference to an array of floats, and count now refers to the number of floating point numbers. The data block received from the device is an array of 32-bit IEEE 754 format floating point numbers.
Length modifier Z	The data block is assumed to be a reference to an array of doubles, and the count now refers to the number of floating point numbers. The data block received from the device is an array of 64-bit IEEE 754 format floating point numbers.

- **t** Argument type: A reference to a string.

Flags or Modifiers	Interpretation
Default functionality	Characters are read from the device until the first END indicator is received. The character on which the END indicator was received is included in the buffer.
field width	This flag gives the maximum string size. If an END indicator is not received before *field width* number of characters, additional characters are read and discarded until an END indicator arrives. *#field width* has the same meaning as in %s.

- **T** Argument type: A reference to a string.

Flags or Modifiers	Interpretation
Default functionality	Characters are read from the device until the first linefeed character (\n) is received. The linefeed character is included in the buffer.
field width	This flag gives the maximum string size. If a linefeed character is not received before *field width* number of characters, additional characters are read and discarded until a linefeed character arrives. #*field width* has the same meaning as in %s.

- **y** Argument type: A reference to a data array.

Flags or Modifiers	Interpretation
Default functionality	The data block is read as raw binary data. The format specifier sequence should have a flag describing the *array size*, which will give a maximum count of the number of bytes (or words or longwords, depending on length modifiers) to be read from the device. If the *array size* contains a # sign, two arguments are used. The first argument read is a pointer to a long integer specifying the maximum number of elements that the array can hold. The second one should be a reference to an array. Also, in this case the actual number of elements read is stored back in the first argument. In absence of length modifiers, the data is assumed to be of byte-size elements. In some cases, data might be read until an END indicator is read.
Length modifier h	The data block is assumed to be a reference to an array of unsigned short integers (16 bits). The count corresponds to the number of words rather than bytes. If the optional “!ol” byte order modifier is present, the data being read is assumed to be in little endian format; otherwise, the data being read is assumed to be in standard IEEE 488.2 format. Data will be byte swapped and padded as appropriate to native computer format
Length modifier l	The data block is assumed to be a reference to an array of unsigned long integers (32 bits). The count corresponds to the number of longwords rather than bytes. If the optional “!ol” byte order modifier is present, the data being read is assumed to be in little endian format; otherwise, the data being read is assumed to be in standard IEEE 488.2 format. Data will be byte swapped and padded as appropriate to native computer format
Length modifier ll	The data block is assumed to be a reference to an array of unsigned long long integers (64 bits). The count corresponds to the number of longlongwords rather than bytes. If the optional “!ol” byte order modifier is present, the data being read is assumed to be in little endian format; otherwise, the data being read is assumed to be in standard IEEE 488.2 format. Data will be byte swapped and padded as appropriate to native computer format
Byte order modifier !ob	The data being read is assumed to be in standard IEEE 488.2 format. This is the default behavior if neither “!ob” nor “!ol” is present.
Byte order modifier !ol	The data being read is assumed to be in little endian format.

9.1.4.5 BNF Format for viScanf() readFmt String

The following is the BNF format for the `viScanf() readFmt` string:

<scan_fmt> := {<slashed_special> | <conversion> | <ascii_char> } *

<slashed _special> := "\n" | "\r" | "\t" | "\\" | <oct _esc> | "\"

<oct_esc> := "\"<oct_digit> [<oct_digit> [<oct_digit>]]

<ascii_char> := Any ASCII character except slash (\) or percent (%).

<conversion> := <fmt_cod_c> | <fmt_cod_d> | <fmt_cod_e> | <fmt_cod_b> | <fmt_cod_f> | <fmt_cod_s> | <fmt_cod_t> | <fmt_cod_T> | <fmt_cod_y> | "%%"

<fmt_cod_b> := "%" ["*"] [<array_size >] ["h" | "l" | “ll” | "z" | "Z"] "b"

<fmt_cod_c> := "%" ["*"] [<field_width>] "c"

<fmt_cod_d> := "%" ["*"] [","<array_size>] ["l" | “ll” | "h"] "d"

<fmt_cod_e> := "%" ["*"] [","<array_size>] ["l" | "L"] "e"

<fmt_cod_f> := "%" ["*"] [","<array_size>] ["l" | "L"] "f"

<fmt_cod_s> := "%" ["*"] [<field_width>] "s"

<fmt_cod_t> := "%" ["*"] [<field_width>] "t"

<fmt_cod_T> := "%" ["*"] [<field_width>] "T"

<fmt_cod_y> := "%" ["*"] <array_size> [<swap_mod>] ["h" | "l" | “ll”] "y"

<swap_mod> := "!ob" | "!ol"

<field_width> := <positive_integer> | "#"

<array_size> := <positive_integer> | "#"

9.2 API functions

In the following an alphabetical list of VISA functions is presented.

9.2.1 viAssertTrigger

Purpose

Assert software or hardware trigger.

Syntax

```
ViStatus viAssertTrigger(ViSession vi, ViUInt16 protocol)
```

Parameters

vi	IN	Unique logical identifier to session.
protocol	IN	Trigger protocol to use during assertion. Valid values are: VI_TRIG_PROT_DEFAULT, VI_TRIG_PROT_ON, VI_TRIG_PROT_OFF, VI_TRIG_PROT_SYNC, VI_TRIG_PROT_RESERVE, and VI_TRIG_PROT_UNRESERVE.

Return Values

VI_SUCCESS	The specified trigger was successfully asserted to the device.
VI_ERROR_INV_SESSION, VI_ERROR_INV_OBJECT	The given session or object reference is invalid (both are the same value).
VI_ERROR_NSUP_OPER	The given vi does not support this operation.
VI_ERROR_RSRC_LOCKED	Specified operation could not be performed because the resource identified by vi has been locked for this kind of access.
VI_ERROR_INV_PROT	The protocol specified is invalid.
VI_ERROR_TMO	Timeout expired before operation completed.
VI_ERROR_RAW_WR_PROT_VIOL	Violation of raw write protocol occurred during transfer.
VI_ERROR_RAW_RD_PROT_VIOL	Violation of raw read protocol occurred during transfer.
VI_ERROR_INP_PROT_VIOL	Device reported an input protocol error during transfer.
VI_ERROR_BERR	Bus error occurred during transfer.
VI_ERROR_LINE_IN_USE	The specified trigger line is currently in use.
VI_ERROR_NCIC	The interface associated with the given vi is not currently the controller in charge.
VI_ERROR_NLISTENERS	No Listeners condition is detected (both NRFD and NDAC are deasserted).
VI_ERROR_INV_SETUP	Unable to start operation because setup is invalid (due to attributes being set to an inconsistent state).
VI_ERROR_CONN_LOST	The I/O connection for the given session has been lost.
VI_ERROR_LINE_NRESERVED	An attempt was made to use a line that was not reserved.

Description

This operation will source a software or hardware trigger dependent on the interface type. For a GPIB device, the device is addressed to listen, and then the GPIB GET command is sent. For a VXI device, if VI_ATTR_TRIG_ID is VI_TRIG_SW, then the device is

sent the Word Serial Trigger command; for any other values of the attribute, a hardware trigger is sent on the line corresponding to the value of that attribute. For a session to a Serial device or TCP/IP socket, if VI_ATTR_IO_PROT is VI_PROT_4882_STRS, the device is sent the string "*TRG\n"; otherwise, this operation is not valid. For a session to a USB instrument, this function sends the TRIGGER message ID on the Bulk-OUT pipe.

For GPIB, USB, and VXI software triggers, VI_TRIG_PROT_DEFAULT is the only valid protocol. For VXI hardware triggers, VI_TRIG_PROT_DEFAULT is equivalent to VI_TRIG_PROT_SYNC.

9.2.2 viBufRead

Purpose

Similar to viRead(), except that the operation uses the formatted I/O read buffer for holding data read from the device.

Syntax

```
ViStatus viBufRead(ViSession vi, ViPBuf buf, ViUInt32 cnt, ViPUInt32 retCnt)
```

Parameters

vi	IN	Unique logical identifier to a session.
buf	OUT	Represents the location of a buffer to receive data from device.
cnt	IN	Number of bytes to be read.
retCnt	OUT	Represents the location of an integer that will be set to the number of bytes actually transferred.

Return Values

VI_SUCCESS	The operation completed successfully and the END indicator was received (for interfaces that have END indicators).
VI_SUCCESS_TERM_CHAR	The specified termination character was read.
VI_SUCCESS_MAX_CNT	The number of bytes read is equal to count.
VI_ERROR_INV_SESSION, VI_ERROR_INV_OBJECT	The given session or object reference is invalid (both are the same value).
VI_ERROR_NSUP_OPER	The given vi does not support this operation.
VI_ERROR_RSRC_LOCKED	Specified operation could not be performed because the resource identified by vi has been locked for this kind of access.
VI_ERROR_TMO	Timeout expired before operation completed.
VI_ERROR_IO	An unknown I/O error occurred during transfer.

Description

This operation is similar to `viRead()` and does not perform any kind of data formatting. It differs from `viRead()` in that the data is read from the formatted I/O read buffer (the same buffer as used by `viScanf()` and related operations) rather than directly from the device. This operation can intermix with the `viScanf()` operation, but use with the `viRead()` operation is discouraged.

9.2.3 viBufWrite

Purpose

Similar to `viWrite()`, except the data is written to the formatted I/O write buffer rather than directly to the device.

Syntax

```
ViStatus viBufWrite(ViSession vi, ViBuf buf, ViUInt32 cnt, ViPUInt32 retCnt)
```

Parameters

`vi`	IN	Unique logical identifier to a session.
`buf`	IN	Represents the location of a data block to be sent to device.
`cnt`	IN	Specifies number of bytes to be written.
`retCnt`	OUT	Represents the location of an integer that will be set to the number of bytes actually transferred.

Return Values

`VI_SUCCESS`	Operation completed successfully.
`VI_ERROR_INV_SESSION`, `VI_ERROR_INV_OBJECT`	The given session or object reference is invalid (both are the same value).
`VI_ERROR_NSUP_OPER`	The given `vi` does not support this operation.
`VI_ERROR_RSRC_LOCKED`	Specified operation could not be performed because the resource identified by `vi` has been locked for this kind of access.
`VI_ERROR_TMO`	Timeout expired before operation completed.
`VI_ERROR_INV_SETUP`	Unable to start write operation because setup is invalid (due to attributes being set to an inconsistent state).
`VI_ERROR_IO`	An unknown I/O error occurred during transfer.

Description

This operation is similar to `viWrite()` and does not perform any kind of data formatting. It differs from `viWrite()` in that the data is written to the formatted I/O write buffer (the same buffer as used by `viPrintf()` and related operations) rather than directly to the device. This operation can intermix with the `viPrintf()` operation, but mixing it with the `viWrite()` operation is discouraged.

9.2.4 viClear

Purpose

Clear a device.

Syntax

```
ViStatus viClear(ViSession vi)
```

Parameters

vi	IN	Unique logical identifier to a session.

Return Values

VI_SUCCESS	Operation completed successfully.
VI_ERROR_INV_SESSION, VI_ERROR_INV_OBJECT	The given session or object reference is invalid (both are the same value).
VI_ERROR_NSUP_OPER	The given vi does not support this operation.
VI_ERROR_RSRC_LOCKED	Specified operation could not be performed because the resource identified by vi has been locked for this kind of access.
VI_ERROR_TMO	Timeout expired before operation completed.
VI_ERROR_RAW_WR_PROT_VIOL	Violation of raw write protocol occurred during transfer.
VI_ERROR_RAW_RD_PROT_VIOL	Violation of raw read protocol occurred during transfer.
VI_ERROR_BERR	Bus error occurred during transfer.
VI_ERROR_NCIC	The interface associated with the given vi is not currently the controller in charge.
VI_ERROR_NLISTENERS	No Listeners condition is detected (both NRFD and NDAC are deasserted).
VI_ERROR_INV_SETUP	Unable to start operation because setup is invalid (due to attributes being set to an inconsistent state).
VI_ERROR_CONN_LOST	The I/O connection for the given session has been lost.

Description

This operation performs an IEEE 488.1-style clear of the device. For VXI INSTR sessions, VISA must use the Word Serial Clear command. For GPIB INSTR sessions, VISA uses the Selected Device Clear command. For Serial INSTR sessions, VISA must flush (discard) the I/O output buffer, send a break, and then flush (discard) the I/O input buffer. For TCP/IP sessions, VISA must flush (discard) the I/O buffers. Flushing the data may take longer than the VISA timeout without returning VI_ERROR_TMO. For USB INSTR sessions, VISA sends the INITIATE_CLEAR and CHECK_CLEAR_STATUS commands on the control pipe.

9.2.5 viClose

Purpose

Close the specified session, event, or find list.

Syntax

```
ViStatus viClose(ViObject vi)
```

Parameters

vi	IN	Unique logical identifier to a session, event, or find list.

Return Values

VI_SUCCESS	Session, event, or find list closed successfully.
VI_WARN_NULL_OBJECT	The specified object reference is uninitialized.
VI_ERROR_INV_SESSION, VI_ERROR_INV_OBJECT	The given session or object reference is invalid (both are the same value).
VI_ERROR_CLOSING_FAILED	Unable to deallocate the previously allocated data structures corresponding to this session or object reference.

Description

This operation closes a session, event, or a find list. In this process all the data structures that had been allocated for the specified vi are freed

9.2.6 viDisableEvent

Purpose

Disable notification of an event type by the specified mechanisms.

Syntax

```
ViStatus viDisableEvent(ViSession vi, ViEventType eventType, ViUInt16 mechanism)
```

Parameters

vi	IN	Unique logical identifier to a session.
eventType	IN	Logical event identifier.
mechanism	IN	Specifies event handling mechanisms to be disabled. The queuing mechanism is disabled by specifying VI_QUEUE, and the callback mechanism is disabled by specifying VI_HNDLR or VI_SUSPEND_HNDLR. It is possible to disable both mechanisms simultaneously by specifying VI_ALL_MECH.

Return Values

VI_SUCCESS	Event disabled successfully.

VI_SUCCESS_EVENT_DIS	Specified event is already disabled for at least one of the specified mechanisms.
VI_ERROR_INV_SESSION, VI_ERROR_INV_OBJECT	The given session or object reference is invalid (both are the same value).
VI_ERROR_INV_EVENT	Specified event type is not supported by the resource.
VI_ERROR_INV_MECH	Invalid mechanism specified.

Description

This operation disables servicing of an event identified by the eventType parameter for the mechanisms specified in the mechanism parameter. Specifying VI_ALL_ENABLED_EVENTS for the eventType parameter allows a session to stop receiving all events. The session can stop receiving queued events by specifying VI_QUEUE. Applications can stop receiving callback events by specifying either VI_HNDLR or VI_SUSPEND_HNDLR. Specifying VI_ALL_MECH disables both the queuing and callback mechanisms.

9.2.7 viDiscardEvents

Purpose

Discard event occurrences for specified event types and mechanisms in a session.

Syntax

```
ViStatus viDiscardEvents(ViSession vi, ViEventType eventType, ViUInt16 mechanism)
```

Parameters

vi	IN	Unique logical identifier to a session.
eventType	IN	Logical event identifier.
mechanism	IN	Specifies the mechanisms for which the events are to be discarded. The VI_QUEUE value is specified for the queuing mechanism and the VI_SUSPEND_HNDLR value is specified for the pending events in the callback mechanism. It is possible to specify both mechanisms simultaneously by specifying VI_ALL_MECH.

Return Values

VI_SUCCESS	Event queue flushed successfully.
VI_SUCCESS_QUEUE_EMPTY	Operation completed successfully, but queue was empty.
VI_ERROR_INV_SESSION, VI_ERROR_INV_OBJECT	The given session or object reference is invalid (both are the same value).
VI_ERROR_INV_EVENT	Specified event type is not supported by the resource.
VI_ERROR_INV_MECH	Invalid mechanism specified.

Description

This operation discards all pending occurrences of the specified event types and mechanisms from the specified session. The information about all the event occurrences that have not yet been handled is discarded. This operation is useful to remove event occurrences that an application no longer needs.

9.2.8 viEnableEvent

Purpose

Enable notification of a specified event.

Syntax

```
ViStatus viEnableEvent(ViSession vi, ViEventType eventType, ViUInt16 mechanism,
ViEventFilter context)
```

Parameters

vi	IN	Unique logical identifier to a session.
eventType	IN	Logical event identifier.
mechanism	IN	Specifies event handling mechanisms to be enabled. The queuing mechanism is enabled by specifying VI_QUEUE, and the callback mechanism is enabled by specifying VI_HNDLR or VI_SUSPEND_HNDLR. It is possible to enable both mechanisms simultaneously by specifying "bit-wise OR" of VI_QUEUE and one of the two mode values for the callback mechanism.
context	IN	VI_NULL

Return Values

VI_SUCCESS	Event enabled successfully.
VI_SUCCESS_EVENT_EN	Specified event is already enabled for at least one of the specified mechanisms.
VI_ERROR_INV_SESSION, VI_ERROR_INV_OBJECT	The given session or object reference is invalid (both are the same value).
VI_ERROR_INV_EVENT	Specified event type is not supported by the resource.
VI_ERROR_INV_MECH	Invalid mechanism specified.
VI_ERROR_INV_CONTEXT	Specified event context is invalid.
VI_ERROR_HNDLR_NINSTALLED	A handler is not currently installed for the specified event. The session cannot be enabled for the VI_HNDLR mode of the callback mechanism.
VI_ERROR_NSUP_MECH	The specified mechanism is not supported for the given event type.

Description

This operation enables notification of an event identified by the `eventType` parameter for mechanisms specified in the mechanism parameter. The specified session can be enabled to queue events by specifying `VI_QUEUE`. Applications can enable the session to invoke a callback function to execute the handler by specifying `VI_HNDLR`. The applications are required to install at least one handler to be enabled for this mode. Specifying `VI_SUSPEND_HNDLR` enables the session to receive callbacks, but the invocation of the handler is deferred to a later time. Successive calls to this operation replace the old callback mechanism with the new callback mechanism. Specifying `VI_ALL_ENABLED_EVENTS` for the `eventType` parameter refers to all events that have previously been enabled on this session, making it easier to switch between the two callback mechanisms for multiple events.

9.2.9 viFindNext

Purpose

Return the next resource found during a previous call to viFindRsrc().

Syntax

```
ViStatus viFindNext(ViFindList vi, ViChar desc[])
```

Parameters

`vi`	IN	Describes a find list. This parameter must be created by viFindRsrc().
`desc`	OUT	Returns a string identifying the location of a device. Strings can then be passed to viOpen() to establish a session to the given device.

Return Values

`VI_SUCCESS`	Resource(s) found.
`VI_ERROR_INV_SESSION`, `VI_ERROR_INV_OBJECT`	The given session or object reference is invalid (both are the same value).
`VI_ERROR_NSUP_OPER`	The given `findList` does not support this operation.
`VI_ERROR_RSRC_NFOUND`	There are no more matches.

Description

This operation returns the next device found in the list created by `viFindRsrc()`. The list is referenced by the handle that was returned by </c>viFindRsrc()</c>.

9.2.10 viFindRsrc

Purpose

Query a VISA system to locate the resources associated with a specified interface.

Syntax

```
ViStatus viFindRsrc(ViSession sesn, ViString expr, ViPFindList vi, ViPUInt32 retCnt, ViChar desc[])
```

Parameters

sesn	IN	Resource Manager session (should always be the Default Resource Manager for VISA returned from viOpenDefaultRM()).
expr	IN	This is a regular expression followed by an optional logical expression. The grammar for this expression is given below.
vi	OUT	Returns a handle identifying this search session. This handle will be used as an input in viFindNext().
retcnt	OUT	Number of matches.
desc	OUT	Returns a string identifying the location of a device. Strings can then be passed to viOpen() to establish a session to the given device.

Return Values

VI_SUCCESS	Resource(s) found.
VI_ERROR_INV_SESSION, VI_ERROR_INV_OBJECT	The given session or object reference is invalid (both are the same value).
VI_ERROR_NSUP_OPER	The given sesn does not support this operation.
VI_ERROR_INV_EXPR	Invalid expression specified for search.
VI_ERROR_RSRC_NFOUND	Specified expression does not match any devices.

Description

This operation matches the value specified in the expr parameter with the resources available for a particular interface. On successful completion, it returns the first resource found in the list and returns a count to indicate if there were more resources found for the designated interface. This function also returns the handle vi to a find list. This handle points to the list of resources and it must be used as an input to viFindNext(). When this handle is no longer needed, it should be passed to viClose().

9.2.11 viFlush

Purpose

Manually flush the specified buffers associated with formatted I/O operations and/or serial communication.

Syntax

```
ViStatus viFlush(ViSession vi, ViUInt16 mask)
```

Parameters

vi	IN	Unique logical identifier to a session.
mask	IN	Specifies the action to be taken with flushing the buffer.

Return Values

VI_SUCCESS	Buffers flushed successfully.
VI_ERROR_INV_SESSION, VI_ERROR_INV_OBJECT	The given session or object reference is invalid (both are the same value).
VI_ERROR_RSRC_LOCKED	Specified operation could not be performed because the resource identified by vi has been locked for this kind of access.
VI_ERROR_IO	Could not perform read/write operation because of I/O error.
VI_ERROR_TMO	The read/write operation was aborted because timeout expired while operation was in progress.
VI_ERROR_INV_MASK	The specified mask does not specify a valid flush operation on read/write resource.
VI_READ_BUF	Discard the read buffer contents and if data was present in the read buffer and no END-indicator was present, read from the device until encountering an END indicator (which causes the loss of data). This action resynchronizes the next viScanf() call to read a <TERMINATED RESPONSE MESSAGE>. (Refer to the IEEE 488.2 standard.)
VI_READ_BUF_DISCARD	Discard the read buffer contents (does not perform any I/O to the device).
VI_WRITE_BUF	Flush the write buffer by writing all buffered data to the device.
VI_WRITE_BUF_DISCARD	Discard the write buffer contents (does not perform any I/O to the device).
VI_IO_IN_BUF	Discards the receive buffer contents (same as VI_IO_IN_BUF_DISCARD).
VI_IO_IN_BUF_DISCARD	Discard the receive buffer contents (does not perform any I/O to the device).
VI_IO_OUT_BUF	Flush the transmit buffer by writing all buffered data to the device.
VI_IO_OUT_BUF_DISCARD	Discard the transmit buffer contents (does not perform any I/O to the device).

Description

The value of `mask` can be one of the following flags:

- `VI_READ_BUF`: Discard the read buffer contents and if data was present in the read buffer and no END-indicator was present, read from the device until encountering an END indicator (which causes the loss of data). This action resynchronizes the next `viScanf()` call to read a <TERMINATED RESPONSE MESSAGE>. (Refer to the IEEE 488.2 standard.)
- `VI_READ_BUF_DISCARD`: Discard the read buffer contents (does not perform any I/O to the device).
- `VI_WRITE_BUF`: Flush the write buffer by writing all buffered data to the device.
- `VI_WRITE_BUF_DISCARD`: Discard the write buffer contents (does not perform any I/O to the device).
- `VI_IO_IN_BUF`: Discards the receive buffer contents (same as `VI_IO_IN_BUF_DISCARD`).
- `VI_IO_IN_BUF_DISCARD`: Discard the receive buffer contents (does not perform any I/O to the device).
- `VI_IO_OUT_BUF`: Flush the transmit buffer by writing all buffered data to the device.
- `VI_IO_OUT_BUF_DISCARD`: Discard the transmit buffer contents (does not perform any I/O to the device).

It is possible to combine any of these read flags and write flags for different buffers by OR-ing the flags. However, combining two flags for the same buffer in the same call to `viFlush()` is illegal.

9.2.12 viGetAttribute

Purpose

Retrieve the state of an attribute.

Syntax

```
ViStatus viGetAttribute(ViObject vi, ViAttr attrName, void _VI_PTR attrValue)
```

Parameters

`vi`	IN	Unique logical identifier to a session, event, or find list.
`attrName`	IN	Session, event, or find list attribute for which the state query is made.
`attrValue`	OUT	The state of the queried attribute for a specified resource. The interpretation of the returned value is defined by the individual resource.

Return Values

VI_SUCCESS	Session, event, or find list attribute retrieved successfully.
VI_ERROR_INV_SESSION, VI_ERROR_INV_OBJECT	The given session or object reference is invalid (both are the same value).
VI_ERROR_NSUP_ATTR	The specified attribute is not defined by the referenced session, event, or find list.

Description

The viGetAttribute() operation is used to retrieve the state of an attribute for the specified session, event, or find list.

9.2.13 viGpibCommand

Purpose

Write GPIB command bytes on the bus.

Syntax

```
ViStatus viGpibCommand(ViSession vi, ViBuf cmd, ViUInt32 cnt, ViPUInt32 retCnt)
```

Parameters

vi	IN	Unique logical identifier to a session.
cmd	IN	Buffer containing valid GPIB commands.
cnt	IN	Number of bytes to be written.
retCount	OUT	Number of bytes actually transferred.

Return Values

VI_SUCCESS	Operation completed successfully.
VI_ERROR_INV_SESSION, VI_ERROR_INV_OBJECT	The given session or object reference is invalid (both are the same value).
VI_ERROR_NSUP_OPER	The given vi does not support this operation.
VI_ERROR_RSRC_LOCKED	Specified operation could not be performed because the resource identified by vi has been locked for this kind of access.
VI_ERROR_TMO	Timeout expired before operation completed.
VI_ERROR_INV_SETUP	Unable to start write operation because setup is invalid (due to attributes being set to an inconsistent state).
VI_ERROR_NCIC	The interface associated with the given vi is not currently the controller in charge.
VI_ERROR_NLISTENERS	No Listeners condition is detected (both NRFD and NDAC are deasserted).
VI_ERROR_IO	An unknown I/O error occurred during transfer.

Description

This operation attempts to write count number of bytes of GPIB commands to the interface bus specified by vi. This operation is valid only on GPIB INTFC (interface) sessions. This operation returns only when the transfer terminates.

9.2.14 viGpibControlATN

Purpose

Controls the state of the GPIB ATN interface line, and optionally the active controller state of the local interface board.

Syntax

```
ViStatus viGpibControlATN(ViSession vi, ViUInt16 mode)
```

Parameters

`vi`	IN	Unique logical identifier to a session.
`mode`	IN	Specifies the state of the ATN line and optionally the local active controller state. See the Description section for actual values.

Return Values

`VI_SUCCESS`	Operation completed successfully.
`VI_ERROR_INV_SESSION`, `VI_ERROR_INV_OBJECT`	The given session or object reference is invalid (both are the same value).
`VI_ERROR_NSUP_OPER`	The given `vi` does not support this operation.
`VI_ERROR_RSRC_LOCKED`	Specified operation could not be performed because the resource identified by `vi` has been locked for this kind of access.
`VI_ERROR_NCIC`	The interface associated with this session is not currently the controller in charge.
`VI_ERROR_INV_MODE`	The value specified by the `mode` parameter is invalid.
`VI_ERROR_NSUP_MODE`	The specified mode is not supported by this VISA implementation.

Description

This operation asserts or deasserts the GPIB ATN interface line according to the specified mode. The mode can also specify whether the local interface board should acquire or release Controller Active status. This operation is valid only on GPIB INTFC (interface) sessions.

It is generally not necessary to use the `viGpibControlATN()` operation in most applications. Other operations such as `viGpibCommand()` and `viGpibPassControl()` modify the ATN and/or CIC state automatically. The following modes are available:

– `VI_GPIB_ATN_DEASSERT` Deassert ATN line.

- VI_GPIB_ATN_ASSERT Assert ATN line synchronously (in 488 terminology). If a data handshake is in progress, ATN will not be asserted until the handshake is complete.
- VI_GPIB_ATN_DEASSERT_HANDSHAKE Deassert ATN line, and enter shadow handshake mode. The local board will participate in data handshakes as an Acceptor without actually reading the data.
- VI_GPIB_ATN_ASSERT_IMMEDIATE Assert ATN line asynchronously (in 488 terminology). This should generally be used only under error conditions.

9.2.15 viGpibControlREN

Purpose

Controls the state of the GPIB REN interface line, and optionally the remote/local state of the device.

Syntax

```
ViStatus viGpibControlREN(ViSession vi, ViUInt16 mode)
```

Parameters

vi	IN	Unique logical identifier to a session.
mode	IN	Specifies the state of the REN line and optionally the device remote/local state. See the Description section for actual values.

Return Values

VI_SUCCESS	Operation completed successfully.
VI_ERROR_INV_SESSION, VI_ERROR_INV_OBJECT	The given session or object reference is invalid (both are the same value).
VI_ERROR_NSUP_OPER	The given vi does not support this operation.
VI_ERROR_RSRC_LOCKED	Specified operation could not be performed because the resource identified by vi has been locked for this kind of access.
VI_ERROR_NCIC	The interface associated with this session is not currently the controller in charge.
VI_ERROR_NLISTENERS	No listeners condition is detected (both NRFD and NDAC are deasserted).
VI_ERROR_NSYS_CNTLR	The interface associated with this session is not the system controller.
VI_ERROR_INV_MODE	The value specified by the mode parameter is invalid.

Description

This operation asserts or deasserts the GPIB REN interface line according to the specified `mode`. The `mode` can also specify whether the device associated with this session should be placed in local state (before deasserting REN) or remote state (after asserting REN). This operation is valid only if the GPIB interface associated with the session specified by `vi` is currently the system controller. The following modes are available:

- `VI_GPIB_REN_DEASSERT`: Deassert REN line.
- `VI_GPIB_REN_ASSERT`: Assert REN line.
- `VI_GPIB_REN_DEASSERT_GTL`: Send the Go To Local command (GTL) to this device and deassert REN line.
- `VI_GPIB_REN_ASSERT_ADDRESS`: Assert REN line and address this device.
- `VI_GPIB_REN_ASSERT_LLO`: Send LLO to any devices that are addressed to listen.
- `VI_GPIB_REN_ASSERT_ADDRESS_LLO`: Address this device and send it LLO, putting it in RWLS.
- `VI_GPIB_REN_ADDRESS_GTL`: Send the Go To Local command (GTL) to this device.

9.2.16 viGpibPassControl

Purpose

Tell the GPIB device at the specified address to become controller in charge (CIC).

Syntax

```
ViStatus viGpibPassControl(ViSession vi, ViUInt16 primAddr, ViUInt16 secAddr)
```

Parameters

`vi`	IN	Unique logical identifier to a session.
`primAddr`	IN	Primary address of the GPIB device to which you want to pass control.
`secAddr`	IN	Secondary address of the targeted GPIB device. If the targeted device does not have a secondary address, this parameter should contain the value `VI_NO_SEC_ADDR`.

Return Values

`VI_SUCCESS`	Operation completed successfully.
`VI_ERROR_INV_SESSION`, `VI_ERROR_INV_OBJECT`	The given session or object reference is invalid (both are the same value).
`VI_ERROR_NSUP_OPER`	The given `vi` does not support this operation.
`VI_ERROR_RSRC_LOCKED`	Specified operation could not be performed because the resource identified by `vi` has been locked for this kind of access.

VI_ERROR_TMO	Timeout expired before operation completed.
VI_ERROR_NCIC	The interface associated with the given vi is not currently the controller in charge.
VI_ERROR_NLISTENERS	No Listeners condition is detected (both NRFD and NDAC are deasserted).
VI_ERROR_IO	An unknown I/O error occurred during transfer.

Description

This operation passes controller in charge status to the device indicated by primAddr and secAddr, and then deasserts the ATN line. This operation assumes that the targeted device has controller capability. This operation is valid only on GPIB INTFC (interface) sessions.

9.2.17 viGpibSendIFC

Purpose

Pulse the interface clear line (IFC) for at least 100 micorseconds.

Syntax

```
ViStatus viGpibSendIFC(ViSession vi)
```

Parameters

vi	IN	Unique logical identifier to a session.

Return Values

VI_SUCCESS	Operation completed successfully.
VI_ERROR_INV_SESSION, VI_ERROR_INV_OBJECT	The given session or object reference is invalid (both are the same value).
VI_ERROR_NSUP_OPER	The given vi does not support this operation.
VI_ERROR_RSRC_LOCKED	Specified operation could not be performed because the resource identified by vi has been locked for this kind of access.
VI_ERROR_NSYS_CNTLR	The interface associated with this session is not the system controller.

Description

This operation asserts the IFC line and becomes controller in charge (CIC). The local board must be the system controller. This operation is valid only on GPIB INTFC (interface) sessions.

9.2.18 viInstallHandler

Purpose

Install handlers for event callbacks.

Syntax

```
ViStatus viInstallHandler(ViSession vi, ViEventType eventType, ViHndlr handler, ViAddr userHandle)
```

Parameters

vi	IN	Unique logical identifier to a session.
eventType	IN	Logical event identifier.
handler	IN	Interpreted as a valid reference to a handler to be installed by a client application.
userHandle	IN	A value specified by an application that can be used for identifying handlers uniquely for an event type.

Return Values

VI_SUCCESS	Event handler installed successfully.
VI_ERROR_INV_SESSION, VI_ERROR_INV_OBJECT	The given session or object reference is invalid (both are the same value).
VI_ERROR_INV_EVENT	Specified event type is not supported by the resource.
VI_ERROR_INV_HNDLR_REF	The given handler reference is invalid.
VI_ERROR_HNDLR_NINSTALLED	The handler was not installed. This may be returned if an application attempts to install multiple handlers for the same event on the same session.

Description

This operation allows applications to install handlers on sessions. The handler specified in the `handler` parameter is installed along with previously installed handlers for the specified event. Applications can specify a value in the `userHandle` parameter that is passed to the handler on its invocation. VISA identifies handlers uniquely using the handler reference and this value.

9.2.19 viLock

Purpose

Establish an access mode to the specified resource.

Syntax

```
ViStatus viLock(ViSession vi, ViAccessMode lockType, ViUInt32 timeout, ViKeyId requestedKey, ViChar accessKey[])
```

Parameters

vi	IN	Unique logical identifier to a session.
lockType	IN	Specifies the type of lock requested, which can be either `VI_EXCLUSIVE_LOCK` or `VI_SHARED_LOCK`.

timeout	IN	Absolute time period (in milliseconds) that a resource waits to get unlocked by the locking session before returning this operation with an error.
requestedKey	IN	This parameter is not used and should be set to VI_NULL when lockType is VI_EXCLUSIVE_LOCK (exclusive locks). When trying to lock the resource as VI_SHARED_LOCK (shared), a session can either set it to VI_NULL, so that VISA generates an accessKey for the session, or the session can suggest an accessKey to use for the shared lock. Refer to the description section below for more details.
accessKey	OUT	This parameter should be set to VI_NULL when lockType is VI_EXCLUSIVE_LOCK (exclusive locks). When trying to lock the resource as VI_SHARED_LOCK (shared), the resource returns a unique access key for the lock if the operation succeeds. This accessKey can then be passed to other sessions to share the lock.

Return Values

VI_SUCCESS	Specified access mode is successfully acquired.
VI_SUCCESS_NESTED_EXCLUSIVE	Specified access mode is successfully acquired, and this session has nested exclusive locks.
VI_SUCCESS_NESTED_SHARED	Specified access mode is successfully acquired, and this session has nested shared locks.
VI_ERROR_INV_SESSION, VI_ERROR_INV_OBJECT	The given session or object reference is invalid (both are the same value).
VI_ERROR_RSRC_LOCKED	Specified type of lock cannot be obtained because the resource is already locked with a lock type incompatible with the lock requested.
VI_ERROR_INV_LOCK_TYPE	The specified type of lock is not supported by this resource.
VI_ERROR_INV_ACCESS_KEY	The requestedKey value passed in is not a valid access key to the specified resource.
VI_ERROR_TMO	Specified type of lock could not be obtained within the specified timeout period.

Description

This operation is used to obtain a lock on the specified resource. The caller can specify the type of lock requested—exclusive or shared lock—and the length of time the operation will suspend while waiting to acquire the lock before timing out. This operation can also be used for sharing and nesting locks.

The requestedKey and the accessKey parameters apply only to shared locks. These parameters are not applicable when using the lock type VI_EXCLUSIVE_LOCK; in this case, requestedKey and accessKey should be set to VI_NULL. VISA allows user applications to

specify a key to be used for lock sharing, through the use of the requestedKey parameter. Alternatively, a user application can pass VI_NULL for the requestedKey parameter when obtaining a shared lock, in which case VISA will generate a unique access key and return it through the accessKey parameter. If a user application does specify a requestedKey value, VISA will try to use this value for the accessKey. As long as the resource is not locked, VISA will use the requestedKey as the access key and grant the lock. When the operation succeeds, the requestedKey will be copied into the user buffer referred to by the accessKey parameter.

The session that gained a shared lock can pass the accessKey to other sessions for the purpose of the sharing the lock. The session wanting to join the group of sessions sharing the lock can use the key as an input value to the requestedKey parameter. VISA will add the session to the list of sessions sharing the lock, as long as the requestedKey value matches the accessKey value for the particular resource. The session obtaining a shared lock in this manner will then have the same access privileges as the original session that obtained the lock.

It is also possible to obtain nested locks through this operation. To acquire nested locks, invoke the viLock() operation with the same lock type as the previous invocation of this operation. For each session, viLock() and viUnlock() share a lock count, which is initialized to 0. Each invocation of viLock() for the same session (and for the same lockType) increases the lock count. In the case of a shared lock, it returns with the same accessKey every time. When a session locks the resource a multiple number of times, it is necessary to invoke the viUnlock() operation an equal number of times in order to unlock the resource. That is, the lock count increments for each invocation of viLock(), and decrements for each invocation of viUnlock(). A resource is actually unlocked only when the lock count is 0.

9.2.20 viOpen

Purpose

Open a session to the specified device.

Syntax

```
ViStatus viOpen(ViSession sesn, ViRsrc name, ViAccessMode mode, ViUInt32 timeout,
ViPSession vi)
```

Parameters

sesn	IN	Resource Manager session (should always be the Default Resource Manager for VISA returned from viOpenDefaultRM()).
name	IN	Unique symbolic name of a resource.
mode	IN	Specifies the modes by which the resource is to be accessed. The value VI_EXCLUSIVE_LOCK is used to acquire an exclusive lock immediately upon opening a session; if a lock cannot be acquired, the session is closed and an error is returned. The value VI_LOAD_CONFIG is used to configure attributes to values

		specified by some external configuration utility; if this value is not used, the session uses the default values provided by this specification. Multiple access modes can be used simultaneously by specifying a "bit-wise OR" of the above values.
timeout	IN	If the accessMode parameter requests a lock, then this parameter specifies the absolute time period (in milliseconds) that the resource waits to get unlocked before this operation returns an error.
vi	OUT	Unique logical identifier reference to a session.

Return Values

VI_SUCCESS	Session opened successfully.
VI_SUCCESS_DEV_NPRESENT	Session opened successfully, but the device at the specified address is not responding.
VI_WARN_CONFIG_NLOADED	The specified configuration either does not exist or could not be loaded; using VISA-specified defaults.
VI_ERROR_INV_SESSION, VI_ERROR_INV_OBJECT	The given session or object reference is invalid (both are the same value).
VI_ERROR_NSUP_OPER	The given sesn does not support this operation. For VISA, this operation is supported only by the Default Resource Manager session.
VI_ERROR_INV_RSRC_NAME	Invalid resource reference specified. Parsing error.
VI_ERROR_INV_ACC_MODE	Invalid access mode.
VI_ERROR_RSRC_NFOUND	Insufficient location information or resource not present in the system.
VI_ERROR_ALLOC	Insufficient system resources to open a session.
VI_ERROR_RSRC_BUSY	The resource is valid, but VISA cannot currently access it.
VI_ERROR_RSRC_LOCKED	Specified type of lock cannot be obtained because the resource is already locked with a lock type incompatible with the lock requested.
VI_ERROR_TMO	A session to the resource could not be obtained within the specified timeout period.
VI_ERROR_LIBRARY_NFOUND	A code library required by VISA could not be located or loaded.
VI_ERROR_INTF_NUM_NCONFIG	The interface type is valid but the specified interface number is not configured.

Description

This operation opens a session to the specified device. It returns a session identifier that can be used to call any other operations of that device.

9.2.21 viOpenDefaultRM

Purpose

Return a session to the Default Resource Manager resource.

Syntax

```
ViStatus viOpenDefaultRM(ViPSession vi)
```

Parameters

`vi`	OUT	Unique logical identifier to a Default Resource Manager session.

Return Values

`VI_SUCCESS`	Session to the Default Resource Manager resource created successfully.
`VI_ERROR_SYSTEM_ERROR`	The VISA system failed to initialize.
`VI_ERROR_ALLOC`	Insufficient system resources to create a session to the Default Resource Manager resource.
`VI_ERROR_INV_SETUP`	Some implementation-specific configuration file is corrupt or does not exist.
`VI_ERROR_LIBRARY_NFOUND`	A code library required by VISA could not be located or loaded.

Description

This function must be called before any VISA operations can be invoked. The first call to this function initializes the VISA system, including the Default Resource Manager resource, and also returns a session to that resource. Subsequent calls to this function return unique sessions to the same Default Resource Manager resource.

9.2.22 viParseRsrc

Purpose

Parse a resource string to get the interface information.

Syntax

```
ViStatus viParseRsrc(ViSession rmSesn, ViRsrc rsrcName, ViPUInt16 intfType,
ViPUInt16 intfNum)
```

Parameters

`rmSesn`	IN	Resource Manager session (should always be the Default Resource Manager for VISA returned from viOpenDefaultRM()).

rsrcName	IN	Unique symbolic name of a resource.
intfType	OUT	Interface type of the given resource string.
intfNum	OUT	Board number of the interface of the given resource string.

Description

This operation parses a resource string to verify its validity. It should succeed for all strings returned by viFindRsrc() and recognized by viOpen(). This operation is useful if you want to know what interface a given resource descriptor would use without actually opening a session to it.

The values returned in intfType and intfNum correspond to the attributes VI_ATTR_INTF_TYPE and VI_ATTR_INTF_NUM. These values would be the same if a user opened that resource with viOpen() and queried the attributes with viGetAttribute().

9.2.23 viParseRsrcEx

Purpose

Parse a resource string to get extended interface information.

Syntax

```
ViStatus viParseRsrcEx(ViSession rmSesn, ViRsrc rsrcName, ViPUInt16 intfType, ViPUInt16 intfNum, ViChar rsrcClass[], ViChar expandedUnaliasedName[], ViChar aliasIfExists[])
```

Parameters

rmSesn	IN	Resource Manager session (should always be the Default Resource Manager for VISA returned from viOpenDefaultRM()).
rsrcName	IN	Unique symbolic name of a resource.
intfType	OUT	Interface type of the given resource string.
intfNum	OUT	Board number of the interface of the given resource string.
rsrcClass	OUT	Specifies the resource class (for example, “INSTR”) of the given resource string.
expandedUnaliasedName	OUT	This is the expanded version of the given resource string. The format should be similar to the VISA-defined canonical resource name.
aliasIfExists	OUT	Specifies the user-defined alias for the given resource string.

Return Values

VI_SUCCESS	Resource string is valid.
VI_WARN_EXT_FUNC_NIMPL	The operation succeeded, but a lower level driver did not implement the extended functionality.
VI_ERROR_INV_SESSION, VI_ERROR_INV_OBJECT	The given session or object reference is invalid (both are the same value).
VI_ERROR_NSUP_OPER	The given sesn does not support this operation. For VISA, this operation is supported only by the Default Resource Manager session.
VI_ERROR_INV_RSRC_NAME	Invalid resource reference specified. Parsing error.
VI_ERROR_RSRC_NFOUND	Insufficient location information or resource not present in the system.
VI_ERROR_ALLOC	Insufficient system resources to parse the string.
VI_ERROR_LIBRARY_NFOUND	A code library required by VISA could not be located or loaded.
VI_ERROR_INTF_NUM_NCONFIG	The interface type is valid but the specified interface number is not configured.

Description

This operation parses a resource string to verify its validity. It should succeed for all strings returned by viFindRsrc() and recognized by viOpen(). This operation is useful if you want to know what interface a given resource descriptor would use without actually opening a session to it.

The values returned in intfType, intfNum, and rsrcClass correspond to the attributes VI_ATTR_INTF_TYPE, VI_ATTR_INTF_NUM, and VI_ATTR_RSRC_CLASS. These values would be the same if a user opened that resource with viOpen() and queried the attributes with viGetAttribute().

The value returned in expandedUnaliasedName should in most cases be identical to the VISA-defined canonical resource name. However, there may be cases where the canonical name includes information that the driver may not know until the resource has actually been opened. In these cases, the value returned in this parameter must be semantically similar.

The value returned in aliasIfExists allows programmatic access to user-defined aliases. If multiple aliases for a single resource are defined one alias is picked.

9.2.24 viPrintf

Purpose

Convert, format, and send the parameters arg1, arg2, … to the device as specified by the format string (cf. Sec. 9.1.3).

Syntax

```
ViStatus viPrintf(ViSession vi, ViString writeFmt, arg1, arg2, ...)
```

Parameters

vi	IN	Unique logical identifier to a session.
writeFmt	IN	String describing the format for arguments.
arg1, arg2, …	IN	Parameters format string is applied to.

Return Values

VI_SUCCESS	Parameters were successfully formatted.
VI_ERROR_INV_SESSION, VI_ERROR_INV_OBJECT	The given session or object reference is invalid (both are the same value).
VI_ERROR_RSRC_LOCKED	Specified operation could not be performed because the resource identified by vi has been locked for this kind of access.
VI_ERROR_IO	Could not perform write operation because of I/O error.
VI_ERROR_TMO	Timeout expired before write operation completed.
VI_ERROR_INV_FMT	A format specifier in the writeFmt string is invalid.
VI_ERROR_NSUP_FMT	A format specifier in the writeFmt string is not supported.
VI_ERROR_ALLOC	The system could not allocate a formatted I/O buffer because of insufficient system resources.

Description

This operation sends data to a device as specified by the format string. Before sending the data, the operation formats the arg characters in the parameter list as specified in the writeFmt string. The viWrite() operation performs the actual low-level I/O to the device. As a result, you should not use the viWrite() and viPrintf() operations in the same session. The writeFmt string follows the ANSI C format rules for printf.

9.2.25 viQueryf

Purpose

Perform a formatted write and read through a single operation invocation.

Syntax

```
ViStatus viQueryf(ViSession vi, ViString writeFmt, ViString readFmt, arg1, arg2, ...)
```

Parameters

vi	IN	Unique logical identifier to a session.
writeFmt	IN	ViString describing the format of write arguments.
readFmt	IN	ViString describing the format of read arguments.
arg1, arg2, ...	IN OUT	Parameters on which write and read format strings are applied.

Return Values

VI_SUCCESS	Successfully completed the Query operation.
VI_ERROR_INV_SESSION, VI_ERROR_INV_OBJECT	The given session or object reference is invalid (both are the same value).
VI_ERROR_RSRC_LOCKED	Specified operation could not be performed because the resource identified by vi has been locked for this kind of access.
VI_ERROR_IO	Could not perform read/write operation because of I/O error.
VI_ERROR_TMO	Timeout occurred before read/write operation completed.
VI_ERROR_INV_FMT	A format specifier in the writeFmt or readFmt string is invalid.
VI_ERROR_NSUP_FMT	The format specifier is not supported for current argument type.
VI_ERROR_ALLOC	The system could not allocate a formatted I/O buffer because of insufficient system resources.

Description

This operation provides a mechanism of "Send, then receive" typical to a command sequence from a commander device. In this manner, the response generated from the command can be read immediately.

This operation is a combination of the viPrintf() and viScanf() operations. The first n arguments corresponding to the first format string are formatted by using the writeFmt string and then sent to the device. The write buffer is flushed immediately after the write portion of the operation completes. After these actions, the response data is read from the device into the remaining parameters (starting from parameter n+1) using the readFmt string.

This operation returns the same VISA status codes as viPrintf(), viScanf(), and viFlush().

9.2.26 viRead

Purpose

Read data from device synchronously.

Syntax

```
ViStatus viRead(ViSession vi, ViPBuf buf, ViUInt32 cnt, ViPUInt32 retCnt)
```

Parameters

vi	IN	Unique logical identifier to a session.
buf	OUT	Represents the location of a buffer to receive data from device.
cnt	IN	Number of bytes to be read.

retCnt	OUT	Represents the location of an integer that will be set to the number of bytes actually transferred.

Return Values

VI_SUCCESS	The operation completed successfully and the END indicator was received (for interfaces that have END indicators).
VI_SUCCESS_TERM_CHAR	The specified termination character was read.
VI_SUCCESS_MAX_CNT	The number of bytes read is equal to count.
VI_ERROR_INV_SESSION, VI_ERROR_INV_OBJECT	The given session or object reference is invalid (both are the same value).
VI_ERROR_NSUP_OPER	The given vi does not support this operation.
VI_ERROR_RSRC_LOCKED	Specified operation could not be performed because the resource identified by vi has been locked for this kind of access.
VI_ERROR_TMO	Timeout expired before operation completed.
VI_ERROR_RAW_WR_PROT_VIOL	Violation of raw write protocol occurred during transfer.
VI_ERROR_RAW_RD_PROT_VIOL	Violation of raw read protocol occurred during transfer.
VI_ERROR_OUTP_PROT_VIOL	Device reported an output protocol error during transfer.
VI_ERROR_BERR	Bus error occurred during transfer.
VI_ERROR_INV_SETUP	Unable to start read operation because setup is invalid (due to attributes being set to an inconsistent state).
VI_ERROR_NCIC	The interface associated with the given vi is not currently the controller in charge.
VI_ERROR_NLISTENERS	No listeners condition is detected (both NRFD and NDAC are deasserted).
VI_ERROR_ASRL_PARITY	A parity error occurred during transfer.
VI_ERROR_ASRL_FRAMING	A framing error occurred during transfer.
VI_ERROR_ASRL_OVERRUN	An overrun error occurred during transfer. A character was not read from the hardware before the next character arrived.
VI_ERROR_IO	An unknown I/O error occurred during transfer.
VI_ERROR_CONN_LOST	The I/O connection for the given session has been lost.

Description

The read operation synchronously transfers data. The data read is to be stored in the buffer represented by buf. This operation returns only when the transfer terminates. Only one read operation can occur at any one time.

9.2.27 viReadSTB

Purpose

Read a status byte of the service request.

Syntax

```
ViStatus viReadSTB(ViSession vi, ViPUInt16 status)
```

Parameters

vi	IN	Unique logical identifier to the session.
status	OUT	Service request status byte.

Return Values

VI_SUCCESS	Operation completed successfully.
VI_ERROR_INV_SESSION, VI_ERROR_INV_OBJECT	The given session or object reference is invalid (both are the same value).
VI_ERROR_NSUP_OPER	The given vi does not support this operation.
VI_ERROR_RSRC_LOCKED	Specified operation could not be performed because the resource identified by vi has been locked for this kind of access.
VI_ERROR_SRQ_NOCCURRED	Service request has not been received for the session.
VI_ERROR_TMO	Timeout expired before operation completed.
VI_ERROR_RAW_WR_PROT_VIOL	Violation of raw write protocol occurred during transfer.
VI_ERROR_RAW_RD_PROT_VIOL	Violation of raw read protocol occurred during transfer.
VI_ERROR_BERR	Bus error occurred during transfer.
VI_ERROR_NCIC	The interface associated with the given vi is not currently the controller in charge.
VI_ERROR_NLISTENERS	No Listeners condition is detected (both NRFD and NDAC are deasserted).
VI_ERROR_INV_SETUP	Unable to start operation because setup is invalid (due to attributes being set to an inconsistent state).
VI_ERROR_CONN_LOST	The I/O connection for the given session has been lost.

Description

This operation reads a service request status from a service requester (the message-based device). For example, on the IEEE 488.2 interface, the message is read by polling devices; for other types of interfaces, a message is sent in response to a service request to retrieve status information. For a session to a Serial device or TCP/IP socket, if VI_ATTR_IO_PROT is VI_PROT_4882_STRS, the device is sent the string "*STB?\n", and then the device's status byte is read; otherwise, this operation is not valid. If the status information is only one byte long, the most significant byte is returned with the zero value. If the service requester does not respond in the actual timeout period, VI_ERROR_TMO is

returned. For a session to a USB instrument, this function sends the READ_STATUS_BYTE command on the control pipe.

9.2.28 viReadToFile

Purpose

Read data synchronously, and store the transferred data in a file.

Syntax

```
ViStatus viReadToFile(ViSession vi, ViConstString filename, ViUInt32 cnt, ViPUInt32 retCnt)
```

Parameters

vi	IN	Unique logical identifier to a session.
filename	IN	Name of file to which data will be written.
cnt	IN	Number of bytes to be read.
retCnt	OUT	Number of bytes actually transferred.

Return Values

VI_SUCCESS	The operation completed successfully and the END indicator was received (for interfaces that have END indicators).
VI_SUCCESS_TERM_CHAR	The specified termination character was read.
VI_SUCCESS_MAX_CNT	The number of bytes read is equal to count.
VI_ERROR_INV_SESSION, VI_ERROR_INV_OBJECT	The given session or object reference is invalid (both are the same value).
VI_ERROR_NSUP_OPER	The given vi does not support this operation.
VI_ERROR_RSRC_LOCKED	Specified operation could not be performed because the resource identified by vi has been locked for this kind of access.
VI_ERROR_TMO	Timeout expired before operation completed.
VI_ERROR_RAW_WR_PROT_VIOL	Violation of raw write protocol occurred during transfer.
VI_ERROR_RAW_RD_PROT_VIOL	Violation of raw read protocol occurred during transfer.
VI_ERROR_OUTP_PROT_VIOL	Device reported an output protocol error during transfer.
VI_ERROR_BERR	Bus error occurred during transfer.
VI_ERROR_INV_SETUP	Unable to start read operation because setup is invalid (due to attributes being set to an inconsistent state).
VI_ERROR_NCIC	The interface associated with the given vi is not currently the controller in charge.

VI_ERROR_NLISTENERS	No listeners condition is detected (both NRFD and NDAC are deasserted).
VI_ERROR_ASRL_PARITY	A parity error occurred during transfer.
VI_ERROR_ASRL_FRAMING	A framing error occurred during transfer.
VI_ERROR_ASRL_OVERRUN	An overrun error occurred during transfer. A character was not read from the hardware before the next character arrived.
VI_ERROR_IO	An unknown I/O error occurred during transfer.
VI_ERROR_FILE_ACCESS	An error occurred while trying to open the specified file. Possible reasons include an invalid path or lack of access rights.
VI_ERROR_FILE_IO	An error occurred while accessing the specified file.
VI_ERROR_CONN_LOST	The I/O connection for the given session has been lost.

Description

This read operation synchronously transfers data. The file specified in fileName is opened in binary write-only mode. If the value of VI_ATTR_FILE_APPEND_EN is VI_FALSE, any existing contents are destroyed; otherwise, the file contents are preserved. The data read is written to the file. This operation returns only when the transfer terminates.

This operation is useful for storing raw data to be processed later.

9.2.29 viScanf

Purpose

Read, convert, and format data using the format specifier (cf. Sec. 9.1.4). Store the formatted data in the arg1, arg2 parameters.

Syntax

```
ViStatus viScanf(ViSession vi, ViString readFmt, arg1, arg2, ...)
```

Parameters

vi	IN	Unique logical identifier to a session.
readFmt	IN	String describing the format for arguments.
arg1, arg2, …	OUT	A list with the variable number of parameters into which the data is read and the format string is applied.

Return Values

VI_SUCCESS	Data was successfully read and formatted into arg parameter(s).

VI_ERROR_INV_SESSION, VI_ERROR_INV_OBJECT	The given session or object reference is invalid (both are the same value).
VI_ERROR_RSRC_LOCKED	Specified operation could not be performed because the resource identified by vi has been locked for this kind of access.
VI_ERROR_IO	Could not perform read operation because of I/O error.
VI_ERROR_TMO	Timeout expired before read operation completed.
VI_ERROR_INV_FMT	A format specifier in the readFmt string is invalid.
VI_ERROR_NSUP_FMT	A format specifier in the readFmt string is not supported.
VI_ERROR_ALLOC	The system could not allocate a formatted I/O buffer because of insufficient system resources.

Description

This operation receives data from a device, formats it by using the format string, and stores the resultant data in the arg parameter list. The format string can have format specifier sequences, white characters, and ordinary characters. The white characters—blank, vertical tabs, horizontal tabs, form feeds, new line/linefeed, and carriage return—are ignored except in the case of %c and %[ ]. All other ordinary characters except % should match the next character read from the device.

The format string consists of a %, followed by optional modifier flags, followed by one of the format codes in that sequence. It is of the form %[modifier]format code where the optional modifier describes the data format, while format code indicates the nature of data (data type). One and only one format code should be performed at the specifier sequence. A format specification directs the conversion to the next input arg. The results of the conversion are placed in the variable that the corresponding argument points to, unless the * assignment-suppressing character is given. In such a case, no arg is used and the results are ignored.

The viScanf() operation accepts input until an END indicator is read or all the format specifiers in the readFmt string are satisfied. Thus, detecting an END indicator before the readFmt string is fully consumed will result in ignoring the rest of the format string. Also, if some data remains in the buffer after all format specifiers in the readFmt string are satisfied, the data will be kept in the buffer and will be used by the next viScanf operation.

9.2.30 viSetAttribute

Purpose

Set the state of an attribute.

Syntax

```
ViStatus viSetAttribute(ViObject vi, ViAttr attrName, ViAttrState attrValue)
```

Parameters

vi	IN	Unique logical identifier to a session, event, or find list.

attrName	IN	Session, event, or find list attribute for which the state is modified.
attrValue	IN	The state of the attribute to be set for the specified resource. The interpretation of the individual attribute value is defined by the resource.

Return Values

VI_SUCCESS	Attribute value set successfully.
VI_WARN_NSUP_ATTR_STATE	Although the specified attribute state is valid, it is not supported by this implementation.
VI_ERROR_INV_SESSION, VI_ERROR_INV_OBJECT	The given session or object reference is invalid (both are the same value).
VI_ERROR_NSUP_ATTR	The specified attribute is not defined by the referenced session, event, or find list.
VI_ERROR_NSUP_ATTR_STATE	The specified state of the attribute is not valid, or is not supported as defined by the session, event, or find list.
VI_ERROR_ATTR_READONLY	The specified attribute is read-only.
VI_ERROR_RSRC_LOCKED	Specified operation could not be performed because the resource identified by vi has been locked for this kind of access.

Description

The `viSetAttribute()` operation is used to modify the state of an attribute for the specified session, event, or find list.

9.2.31 viSetBuf

Purpose

Set the size for the formatted I/O and/or serial communication buffer(s).

Syntax

```
ViStatus viSetBuf(ViSession vi, ViUInt16 mask, ViUInt32 size)
```

Parameters

vi	IN	Unique logical identifier to a session.
mask	IN	Specifies the type of buffer.
size	IN	The size to be set for the specified buffer(s).

Return Values

VI_SUCCESS	Buffer size set successfully.
VI_WARN_NSUP_BUF	The specified buffer is not supported.
VI_ERROR_INV_SESSION, VI_ERROR_INV_OBJECT	The given session or object reference is invalid (both are the same value).

VI_ERROR_RSRC_LOCKED	Specified operation could not be performed because the resource identified by vi has been locked for this kind of access.
VI_ERROR_ALLOC	The system could not allocate the buffer(s) of the specified size because of insufficient system resources.
VI_ERROR_INV_MASK	The system cannot set the buffer for the given mask.

Description

This operation changes the buffer size of the read and/or write buffer for formatted I/O and/or serial communication. The mask parameter specifies which buffer to set the size of. The mask parameter can specify multiple buffers by bit-ORing any of the following values together: VI_READ_BUF (Formatted I/O read buffer), VI_WRITE_BUF (Formatted I/O write buffer), VI_IO_IN_BUF (I/O communication receive buffer), and VI_IO_OUT_BUF (I/O communication) transmit buffer.

9.2.32 viSPrintf

Purpose

Same as viPrintf(), except the data is written to a user-specified buffer rather than the device (cf. Sec. 9.1.3).

Syntax

```
ViStatus viSPrintf(ViSession vi, ViPBuf buf, ViString writeFmt, arg1, arg2...)
```

Parameters

vi	IN	Unique logical identifier to a session.
buf	OUT	Buffer where data is to be written.
writeFmt	IN	String describing the format for arguments.
arg1, arg2, …	IN	A list containing the variable number of parameters on which the format string is applied. The formatted data is written to the specified device.

Return Values

VI_SUCCESS	Parameters were successfully formatted.
VI_ERROR_INV_SESSION, VI_ERROR_INV_OBJECT	The given session or object reference is invalid (both are the same value).
VI_ERROR_RSRC_LOCKED	Specified operation could not be performed because the resource identified by vi has been locked for this kind of access.
VI_ERROR_INV_FMT	A format specifier in the writeFmt string is invalid.
VI_ERROR_NSUP_FMT	A format specifier in the writeFmt string is not supported.

VI_ERROR_ALLOC	The system could not allocate a formatted I/O buffer because of insufficient system resources.

Description

This operation is similar to `viPrintf()`, except that the output is not written to the device; it is written to the user-specified buffer. This output buffer will be NULL terminated.

9.2.33 viSScanf

Purpose

Same as `viScanf()`, except that the data is read from a user-specified buffer instead of a device (cf. Sec. 9.1.4).

Syntax

```
ViStatus viSScanf(ViSession vi, ViBuf buf, ViString readFmt, arg1, arg2, ...)
```

Parameters

vi	IN	Unique logical identifier to a session.
buf	IN	Buffer from which data is read and formatted.
readFmt	IN	String describing the format for arguments.
arg1, arg2, …	OUT	A list with the variable number of parameters into which the data is read and the format string is applied.

Return Values

VI_SUCCESS	Data was successfully read and formatted into arg parameter(s).
VI_ERROR_INV_SESSION, VI_ERROR_INV_OBJECT	The given session or object reference is invalid (both are the same value).
VI_ERROR_RSRC_LOCKED	Specified operation could not be performed because the resource identified by `vi` has been locked for this kind of access.
VI_ERROR_INV_FMT	A format specifier in the `readFmt` string is invalid.
VI_ERROR_NSUP_FMT	A format specifier in the `readFmt` string is not supported.
VI_ERROR_ALLOC	The system could not allocate a formatted I/O buffer because of insufficient system resources.

Description

This operation is similar to `viScanf()`, except that the data is read from a user-specified buffer rather than a device.

9.2.34 viStatusDesc

Purpose

Return a user-readable description of the status code passed to the operation.

Syntax

```
ViStatus viStatusDesc(ViObject vi, ViStatus status, ViChar desc[])
```

Parameters

vi	IN	Unique logical identifier to a session, event, or find list.
status	IN	Status code to interpret.
desc	OUT	The user-readable string interpretation of the status code passed to the operation.

Return Values

VI_SUCCESS	Description successfully returned.
VI_WARN_UNKNOWN_STATUS	The status code passed to the operation could not be interpreted.

Description

The `viStatusDesc()` operation is used to retrieve a user-readable string that describes the status code presented.

9.2.35 viUninstallHandler

Purpose

Uninstall handlers for events.

Syntax

```
ViStatus viUninstallHandler(ViSession vi, ViEventType eventType, ViHndlr handler,
ViAddr userHandle)
```

Parameters

vi	IN	Unique logical identifier to a session.
eventType	IN	Logical event identifier.
handler	IN	Interpreted as a valid reference to a handler to be uninstalled by a client application.
userHandle	IN	A value specified by an application that can be used for identifying handlers uniquely in a session for an event.

Return Values

VI_SUCCESS	Event handler successfully uninstalled.
VI_ERROR_INV_SESSION, VI_ERROR_INV_OBJECT	The given session or object reference is invalid (both are the same value).
VI_ERROR_INV_EVENT	Specified event type is not supported by the resource.

VI_ERROR_INV_HNDLR_REF	Either the specified handler reference or the user context value (or both) does not match any installed handler.
VI_ERROR_HNDLR_NINSTALLED	A handler is not currently installed for the specified event.

Description

This operation allows client applications to uninstall handlers for events on sessions. Applications should also specify the value in the `userHandle` parameter that was passed while installing the handler. VISA identifies handlers uniquely using the handler reference and this value. All the handlers, for which the handler reference and the value matches, are uninstalled. The following tables list all the VISA-defined values and corresponding actions of uninstalling handlers.

9.2.36 viUnlock

Purpose

Relinquish a lock for the specified resource.

Syntax

```
ViStatus viUnlock(ViSession vi)
```

Parameters

vi	IN	Unique logical identifier to a session.

Return Values

VI_SUCCESS	Lock successfully relinquished.
VI_SUCCESS_NESTED_EXCLUSIVE	Call succeeded, but this session still has nested exclusive locks.
VI_SUCCESS_NESTED_SHARED	Call succeeded, but this session still has nested shared locks.
VI_ERROR_INV_SESSION, VI_ERROR_INV_OBJECT	The given session or object reference is invalid (both are the same value).
VI_ERROR_SESN_NLOCKED	The current session did not have any lock on the resource.

Description

This operation is used to relinquish the lock previously obtained using the `viLock()` operation.

9.2.37 viVPrintf

Purpose

Convert, format, and send `params` to the device as specified by the format string (cf Sec 9.1.3).

Syntax

```
ViStatus viVPrintf(ViSession vi, ViString writeFmt, ViVAList params)
```

Parameters

vi	IN	Unique logical identifier to a session.
writeFmt	IN	The format string to apply to parameters in ViVAList.
params	IN	A list containing the variable number of parameters on which the format string is applied. The formatted data is written to the specified device.

Return Values

VI_SUCCESS	Parameters were successfully formatted.
VI_ERROR_INV_SESSION, VI_ERROR_INV_OBJECT	The given session or object reference is invalid (both are the same value).
VI_ERROR_RSRC_LOCKED	Specified operation could not be performed because the resource identified by vi has been locked for this kind of access.
VI_ERROR_IO	Could not perform write operation because of I/O error.
VI_ERROR_TMO	Timeout expired before write operation completed.
VI_ERROR_INV_FMT	A format specifier in the writeFmt string is invalid.
VI_ERROR_NSUP_FMT	A format specifier in the writeFmt string is not supported.
VI_ERROR_ALLOC	The system could not allocate a formatted I/O buffer because of insufficient system resources.

Description

This operation is similar to viPrintf(), except that the ViVAList parameters list provides the parameters rather than separate arg parameters.

9.2.38 viVQueryf

Purpose

Perform a formatted write and read through a single operation invocation.

Syntax

```
ViStatus viVQueryf(ViSession vi, ViString writeFmt, ViString readFmt, ViVAList params)
```

Parameters

vi	IN	Unique logical identifier to a session.
writeFmt	IN	The format string is applied to write parameters in ViVAList.

readFmt	IN	The format string to applied to read parameters in ViVAList.
params	IN OUT	A list containing the variable number of write and read parameters. The write parameters are formatted and written to the specified device. The read parameters store the data read from the device after the format string is applied to the data.

Return Values

VI_SUCCESS	Successfully completed the Query operation.
VI_ERROR_INV_SESSION, VI_ERROR_INV_OBJECT	The given session or object reference is invalid (both are the same value).
VI_ERROR_RSRC_LOCKED	Specified operation could not be performed because the resource identified by vi has been locked for this kind of access.
VI_ERROR_IO	Could not perform read/write operation because of I/O error.
VI_ERROR_TMO	Timeout occurred before read/write operation completed.
VI_ERROR_INV_FMT	A format specifier in the writeFmt or readFmt string is invalid.
VI_ERROR_NSUP_FMT	The format specifier is not supported for current argument type.
VI_ERROR_ALLOC	The system could not allocate a formatted I/O buffer because of insufficient system resources.

Description

This operation is similar to ViQueryf(), except that the ViVAList parameters list provides the parameters rather than the separate arg parameter list.

9.2.39 viVScanf

Purpose

Read, convert, and format data using the format specifier (cf. Sec. 9.1.4). Store the formatted data in params.

Syntax

```
ViStatus viVScanf(ViSession vi, ViString readFmt, ViVAList params)
```

Parameters

vi	IN	Unique logical identifier to a session.
readFmt	IN	The format string to apply to parameters in ViVAList.

params	OUT	A list with the variable number of parameters into which the data is read and the format string is applied.

Return Values

VI_SUCCESS	Data was successfully read and formatted into params.
VI_ERROR_INV_SESSION, VI_ERROR_INV_OBJECT	The given session or object reference is invalid (both are the same value).
VI_ERROR_RSRC_LOCKED	Specified operation could not be performed because the resource identified by vi has been locked for this kind of access.
VI_ERROR_IO	Could not perform read operation because of I/O error.
VI_ERROR_TMO	Timeout expired before read operation completed.
VI_ERROR_INV_FMT	A format specifier in the readFmt string is invalid.
VI_ERROR_NSUP_FMT	A format specifier in the readFmt string is not supported.
VI_ERROR_ALLOC	The system could not allocate a formatted I/O buffer because of insufficient system resources.

Description

This operation is similar to viScanf(), except that the ViVAList parameters list provides the parameters rather than separate arg parameters.

9.2.40 viVSPrintf

Purpose

Same as viVPrintf(), except that the data is written to a user-specified buffer rather than a device.

Syntax

```
ViStatus viVSPrintf(ViSession vi, ViPBuf buf, ViString writeFmt, ViVAList params)
```

Parameters

vi	IN	Unique logical identifier to a session.
buf	OUT	Buffer where data is to be written.
writeFmt	IN	The format string to apply to parameters in ViVAList.
params	IN	A list containing the variable number of parameters on which the format string is applied. The formatted data is written to the specified device.

Return Values

VI_SUCCESS	Parameters were successfully formatted.

VI_ERROR_INV_SESSION, VI_ERROR_INV_OBJECT	The given session or object reference is invalid (both are the same value).
VI_ERROR_RSRC_LOCKED	Specified operation could not be performed because the resource identified by vi has been locked for this kind of access.
VI_ERROR_INV_FMT	A format specifier in the writeFmt string is invalid.
VI_ERROR_NSUP_FMT	A format specifier in the writeFmt string is not supported.
VI_ERROR_ALLOC	The system could not allocate a formatted I/O buffer because of insufficient system resources.

Description

This operation is similar to viVPrintf(), except that the output is not written to the device; it is written to the user-specified buffer. This output buffer will be NULL terminated.

9.2.41 viVSScanf

Purpose

Same as viVScanf(), except that the data is read from a user-specified buffer instead of a device.

Syntax

```
ViStatus viVSScanf(ViSession vi, ViBuf buf, ViString readFmt, ViVAList params)
```

Parameters

vi	IN	Unique logical identifier to a session.
buf	IN	Buffer from which data is read and formatted.
readFmt	IN	The format string to apply to parameters in ViVAList.
params	OUT	A list with the variable number of parameters into which the data is read and the format string is applied.

Return Values

VI_SUCCESS	Data was successfully read and formatted into params.
VI_ERROR_INV_SESSION, VI_ERROR_INV_OBJECT	The given session or object reference is invalid (both are the same value).
VI_ERROR_RSRC_LOCKED	Specified operation could not be performed because the resource identified by vi has been locked for this kind of access.
VI_ERROR_INV_FMT	A format specifier in the readFmt string is invalid.
VI_ERROR_NSUP_FMT	A format specifier in the readFmt string is not supported.
VI_ERROR_ALLOC	The system could not allocate a formatted I/O buffer because of insufficient system resources.

Description

This operation is similar to `viVScanf()`, except that the data is read from a user-specified buffer rather than a device.

9.2.42 viWaitOnEvent

Purpose

Wait for an occurrence of the specified event for a given session.

Syntax

```
ViStatus viWaitOnEvent(ViSession vi, ViEventType inEventType, ViUInt32 timeout,
ViPEventType outEventType, ViPEvent outContext)
```

Parameters

`vi`	IN	Unique logical identifier to a session.
`inEventType`	IN	Logical identifier of the event(s) to wait for.
`timeout`	IN	Absolute time period in time units that the resource shall wait for a specified event to occur before returning the time elapsed error. The time unit is in milliseconds.
`outEventType`	OUT	Logical identifier of the event actually received.
`outContext`	OUT	A handle specifying the unique occurrence of an event.

Return Values

`VI_SUCCESS`	Wait terminated successfully on receipt of an event occurrence. The queue is empty.
`VI_SUCCESS_QUEUE_NEMPTY`	Wait terminated successfully on receipt of an event notification. There is still at least one more event occurrence of the type specified by `inEventType` available for this session.
`VI_ERROR_INV_SESSION`, `VI_ERROR_INV_OBJECT`	The given session or object reference is invalid (both are the same value).
`VI_ERROR_INV_EVENT`	Specified event type is not supported by the resource.
`VI_ERROR_TMO`	Specified event did not occur within the specified time period.
`VI_ERROR_NENABLED`	The session must be enabled for events of the specified type in order to receive them.

Description

The `viWaitOnEvent()` operation suspends execution of a thread of application and waits for an event `inEventType` for a time period not to exceed that specified by `timeout`. Refer to individual event descriptions for context definitions. If the specified `inEventType` is `VI_ALL_ENABLED_EVENTS`, the operation waits for any event that is enabled for the given session. If the specified `timeout` value is `VI_TMO_INFINITE`, the operation is suspended indefinitely.

9.2.43 viWrite

Purpose

Write data to device synchronously.

Syntax

```
ViStatus viWrite(ViSession vi, ViBuf buf, ViUInt32 cnt, ViPUInt32 retCnt)
```

Parameters

vi	IN	Unique logical identifier to a session.
buf	IN	Represents the location of a data block to be sent to device.
cnt	IN	Specifies number of bytes to be written.
retCnt	OUT	Represents the location of an integer that will be set to the number of bytes actually transferred.

Return Values

VI_SUCCESS	Transfer completed.
VI_ERROR_INV_SESSION, VI_ERROR_INV_OBJECT	The given session or object reference is invalid (both are the same value).
VI_ERROR_NSUP_OPER	The given vi does not support this operation.
VI_ERROR_RSRC_LOCKED	Specified operation could not be performed because the resource identified by vi has been locked for this kind of access.
VI_ERROR_TMO	Timeout expired before operation completed.
VI_ERROR_RAW_WR_PROT_VIOL	Violation of raw write protocol occurred during transfer.
VI_ERROR_RAW_RD_PROT_VIOL	Violation of raw read protocol occurred during transfer.
VI_ERROR_INP_PROT_VIOL	Device reported an input protocol error during transfer.
VI_ERROR_BERR	Bus error occurred during transfer.
VI_ERROR_INV_SETUP	Unable to start write operation because setup is invalid (due to attributes being set to an inconsistent state).
VI_ERROR_NCIC	The interface associated with the given vi is not currently the controller in charge.
VI_ERROR_NLISTENERS	No Listeners condition is detected (both NRFD and NDAC are deasserted).
VI_ERROR_IO	An unknown I/O error occurred during transfer.
VI_ERROR_CONN_LOST	The I/O connection for the given session has been lost.

Description

The write operation synchronously transfers data. The data to be written is in the buffer represented by buf. This operation returns only when the transfer terminates. Only one synchronous write operation can occur at any one time

9.2.44 viWriteFromFile

Purpose

Take data from a file and write it out synchronously.

Syntax

```
ViStatus viWriteFromFile(ViSession vi, ViConstString filename, ViUInt32 cnt,
ViPUInt32 retCnt)
```

Parameters

vi	IN	Unique logical identifier to a session.
fileName	IN	Name of file from which data will be read.
cnt	IN	Number of bytes to be written.
retCnt	OUT	Number of bytes actually transferred.

Return Values

VI_SUCCESS	Transfer completed.
VI_ERROR_INV_SESSION, VI_ERROR_INV_OBJECT	The given session or object reference is invalid (both are the same value).
VI_ERROR_NSUP_OPER	The given vi does not support this operation.
VI_ERROR_RSRC_LOCKED	Specified operation could not be performed because the resource identified by vi has been locked for this kind of access.
VI_ERROR_TMO	Timeout expired before operation completed.
VI_ERROR_RAW_WR_PROT_VIOL	Violation of raw write protocol occurred during transfer.
VI_ERROR_RAW_RD_PROT_VIOL	Violation of raw read protocol occurred during transfer.
VI_ERROR_INP_PROT_VIOL	Device reported an input protocol error during transfer.
VI_ERROR_BERR	Bus error occurred during transfer.
VI_ERROR_NCIC	The interface associated with the given vi is not currently the controller in charge.
VI_ERROR_NLISTENERS	No Listeners condition is detected (both NRFD and NDAC are deasserted).
VI_ERROR_IO	An unknown I/O error occurred during transfer.
VI_ERROR_FILE_ACCESS	An error occurred while trying to open the specified file. Possible reasons include an invalid path or lack of access rights.
VI_ERROR_FILE_IO	An error occurred while accessing the specified file.
VI_ERROR_CONN_LOST	The I/O connection for the given session has been lost.

Description

This write operation synchronously transfers data. The file specified in `fileName` is opened in binary read-only mode, and the data (up to end-of-file or the number of bytes specified in `count`) is read. The data is then written to the device. This operation returns only when the transfer terminates.

This operation is useful for sending data that was already processed and/or formatted.

9.3 Attributes

In the following sections lists of VISA attributes for all available instrument classes are presented.

9.3.1 Instrument class: All

9.3.1.1 VI_ATTR_MAX_QUEUE_LENGTH

Information

R/W Local	`ViUInt32`	1h to FFFFFFFFh

Description

Specifies the maximum number of events that can be queued at any time on the given session.

9.3.1.2 VI_ATTR_RM_SESSION

Information

RO Local	`ViSession`	N/A

Description

Specifies the session of the Resource Manager that was used to open this session.

9.3.1.3 VI_ATTR_RSRC_CLASS

Information

RO Global	`ViString`	N/A

Description

Specifies the resource class (for example, “INSTR”) .

9.3.1.4 VI_ATTR_RSRC_IMPL_VERSION

Information

RO Global	`ViVersion`	0h to FFFFFFFFh

Description

Resource version that uniquely identifies each of the different revisions or implementations of a resource.

9.3.1.5 VI_ATTR_RSRC_LOCK_STATE

Information

RO Global	ViAccessMode	VI_NO_LOCK, VI_EXCLUSIVE_LOCK, VI_SHARED_LOCK

Description

The current locking state of the resource, reflecting any locks granted to an open session to the device using the same interface and protocol. The resource can be unlocked, locked with an exclusive lock, or locked with a shared lock.

9.3.1.6 VI_ATTR_RSRC_MANF_ID

Information

RO Global	ViUInt16	0h to 3FFFh

Description

A value that corresponds to the VXI manufacturer ID of the manufacturer that created the implementation.

9.3.1.7 VI_ATTR_RSRC_MANF_NAME

Information

RO Global	ViString	N/A

Description

A string that corresponds to the VXI manufacturer name of the manufacturer that created the implementation.

9.3.1.8 VI_ATTR_RSRC_NAME

Information

RO Global	ViRsrc	N/A

Description

The unique identifier for a resource.

9.3.1.9 VI_ATTR_RSRC_SPEC_VERSION

Information

RO Global	ViVersion	00500400h

Description

Resource version that uniquely identifies the version of the VISA specification to which the implementation is compliant.

9.3.1.10 VI_ATTR_USER_DATA

Information

R/W Local	ViAddr	N/A

Description

Data used privately by the application for a particular session. This data is not used by VISA for any purposes and is provided to the application for its own use.

9.3.1.11 VI_ATTR_USER_DATA_32

Information

R/W Local	ViUInt32	0h to FFFFFFFFh

Description

Data used privately by the application for a particular session. This data is not used by VISA for any purposes and is provided to the application for its own use.

9.3.1.12 VI_ATTR_USER_DATA_64

Information

R/W Local	ViUInt64	0h to FFFFFFFFFFFFFFFFh

Description

Data used privately by the application for a particular session. This data is not used by VISA for any purposes and is provided to the application for its own use. Defined only for frameworks that are 64-bit native.

9.3.1.13 VI_RS_ATTR_TCPIP_FIND_RSRC_MODE

Information

R/W Global	ViUInt32	VI_RS_FIND_MODE_NONE, VI_RS_FIND_MODE_CONFIG, VI_RS_FIND_MODE_VXI11, VI_RS_FIND_MODE_MDNS

Description

Mode used for discovering devices on the LAN. Different modes may be selected by applying an OR-operation to the desired modes. If VI_RS_FIND_MODE_VXI11 is active devices are found by a VXI-11 broadcast. If VI_RS_FIND_MODE_MDNS is active, devices are found via mDNS/Bonjour.

Note that for this R&S specific attribute the RSVISA_EXTENSION compiler macro has to be defined.

9.3.1.14 VI_RS_ATTR_TCPIP_FIND_RSRC_TMO

Information

R/W Global	ViUInt32	0h to FFFFFFFFh

Description

Timeout for VXI Discovery in Milliseconds.

Note that for this R&S specific attribute the `RSVISA_EXTENSION` compiler macro has to be defined.

9.3.1.15 VI_RS_ATTR_LXI_MANF

Information

RO Global	`ViString`

Description

Manufacturer of the LXI device. This is the first part of a *IDN? query. However, this information is not obtained by a *IDN? query, but from the LXI information provided by the device via the LXI search. Therefore it is only available for devices found by LXI discovery.

Find lists handles returned by viFindRsrc can be queried for this attribute. The value returned corresponds to the last device returned by viFindRsrc or viFindNext, respectively.

Note that for this R&S specific attribute the `RSVISA_EXTENSION` compiler macro has to be defined.

9.3.1.16 VI_RS_ATTR_LXI_MODEL

Information

RO Global	`ViString`

Description

Model name of the LXI device. This is the second part of a *IDN? query. For details see 9.3.1.15.

Note that for this R&S specific attribute the `RSVISA_EXTENSION` compiler macro has to be defined.

9.3.1.17 VI_RS_ATTR_LXI_SERIAL

Information

RO Global	`ViString`

Description

Serial number of the LXI device. This is the third part of a *IDN? query. For details see 9.3.1.15.

Note that for this R&S specific attribute the `RSVISA_EXTENSION` compiler macro has to be defined.

9.3.1.18 VI_RS_ATTR_LXI_VERSION

Information

RO Global	ViString	

Description

Firmware version of the LXI device. This is the fourth part of a *IDN? query. For details see 9.3.1.15.

Note that for this R&S specific attribute the RSVISA_EXTENSION compiler macro has to be defined.

9.3.1.19 VI_RS_ATTR_LXI_DESCRIPTION

Information

RO Global	ViString	

Description

User defined description of the LXI device. For details see 9.3.1.15.

Note that for this R&S specific attribute the RSVISA_EXTENSION compiler macro has to be defined.

9.3.1.20 VI_RS_ATTR_LXI_HOSTNAME

Information

RO Global	ViString	

Description

Hostname of the LXI device. For details see 9.3.1.15.

Note that for this R&S specific attribute the RSVISA_EXTENSION compiler macro has to be defined.

9.3.2 Instrument class: INSTR

9.3.2.1 VI_ATTR_4882_COMPLIANT

Information

RO Global	ViBoolean	VI_TRUE, VI_FALSE

Description

Specifies whether the device is 488.2 compliant.

9.3.2.2 VI_ATTR_ASRL_AVAIL_NUM

Information

RO Global	ViUInt32	0 to FFFFFFFFh

Description

todo

9.3.2.3 VI_ATTR_ASRL_BAUD

Information

RW Global	ViUInt32	0 to FFFFFFFFh

Description

Baud rate of the interface. It is represented as an unsigned 32-bit integer so that any baud rate can be used, but it usually requires a commonly used rate such as 300, 1200, 2400, or 9600 baud.

9.3.2.4 VI_ATTR_ASRL_DATA_BITS

Information

RW Global	ViUInt16	5 to 8

Description

Number of data bits contained in each frame (from 5 to 8). The data bits for each frame are located in the low-order bits of every byte stored in memory.

9.3.2.5 VI_ATTR_ASRL_PARITY

Information

RW Global	ViUInt16	VI_ASRL_PAR_NONE, VI_ASRK_PAR_ODD, VI_ASRL_PAR_EVEN, VI_ASRL_PAR_MARK, VI_ASRL_PAR_SPACE

Description

This is the parity used with every frame transmitted and received. VI_ASRL_PAR_MARK means that the parity bit exists and is always 1. VI_ASRL_PAR_SPACE means that the parity bit exists and is always 0.

9.3.2.6 VI_ATTR_ASRL_STOP_BITS

Information

RW Global	ViUInt16	VI_ASRL_STOP_ONE, VI_ASRL_STOP_ONE5, VI_ASRL_STOP_TWO

Description

This is the number of stop bits used to indicate the end of a frame. The value VI_ASRL_STOP_ONE5 indicates one-and-one-half (1.5) stop bits.

9.3.2.7 VI_ATTR_ASRL_FLOW_CNTRL

Information

RW Global	ViUInt16	VI_ASRL_FLOW_NONE, VI_ASRL_FLOW_XON_XOFF, VI_ASRL_FLOW_RTS_CTS, VI_ASRL_FLOW_DTR_DSR

Description

If this attribute is set to VI_ATTR_ASRL_FLOW_NONE, the transfer mechanism does not use flow control, and buffers on both sides of the connection are assumed to be large enough to hold all data transferred.

If this attribute is set to VI_ATTR_ASRL_FLOW_XON_XOFF, the transfer mechanism uses the *XON* and *XOFF* characters to perform flow control. The transfer mechanism controls input flow by sending *XOFF* when the receive buffer is nearly full, and it controls the output flow by suspending transmission when *XOFF* is received.

If this attribute is set to VI_ATTR_ASRL_FLOW_RTS_CTS, the transfer mechanism uses the *RTS* output signal and the *CTS* input signal to perform flow control. The transfer mechanism controls input flow by unasserting the *RTS* signal when the receive buffer is nearly full, and it controls output flow by suspending the transmission when the *CTS* signal is unasserted.

If this attribute is set to VI_ASRL_FLOW_DTR_DSR, the transfer mechanism uses the DTR output signal and the DSR input signal to perform flow control. The transfer mechanism controls input flow by unasserting the DTR signal when the receive buffer is nearly full, and it controls output flow by suspending the transmission when the DSR signal is unasserted.

This attribute can specify multiple flow control mechanisms by bit-ORing multiple values together. However, certain combinations may not be supported by all serial ports and/or operating systems

9.3.2.8 VI_ATTR_ASRL_END_IN

Information

RW Local	ViUInt16	VI_ASRL_END_NONE, VI_ASRL_END_LAST_BIT, VI_ASRL_END_TERMCHAR

Description

This attribute indicates the method used to terminate read operations. If it is set to VI_ASRL_END_NONE, the read will not terminate until all of the requested data is received (or an error occurs). If it is set to VI_ASRL_END_TERMCHAR, the read will terminate as soon as the character in VI_ATTR_TERMCHAR is received. If it is set to VI_ASRL_END_LAST_BIT, the read will terminate as soon as a character arrives with its last bit set. For example, if VI_ATTR_ASRL_DATA_BITS is set to 8, then the read will terminate when a character arrives with the 8th bit set.

9.3.2.9 VI_ATTR_ASRL_END_OUT

Information

RW Local	ViUInt16	VI_ASRL_END_NONE, VI_ASRL_END_LAST_BIT, VI_ASRL_END_TERMCHAR, ~~VI_ASRL_END_BREAK~~

Description

This attribute indicates the method used to terminate write operations. If it is set to VI_ASRL_END_NONE, the write will not append anything to the data being written. If it is set to VI_ASRL_END_BREAK, the write will transmit a break after all the characters for the write have been sent. If it is set to VI_ASRL_END_LAST_BIT, the write will send all but the last character with the last bit clear, then transmit the last character with the last bit set. For example, if VI_ATTR_ASRL_DATA_BITS is set to 8, then the write will clear the 8th bit for all but the last character, then transmit the last character with the 8th bit

set. If it is set to VI_ASRL_END_TERMCHAR, the write will send the character in VI_ATTR_TERMCHAR after the data being transmitted.

VI_ASRL_END_BREAK is not supported in R&S VISA.

9.3.2.10 VI_ATTR_ASRL_CTS_STATE

Information

RW Global	ViInt16	VI_STATE_ASSERTED, VI_STATE_UNASSERTED, VI_STATE_UNKNOWN

Description

This attribute shows the current state of the Clear To Send (CTS) input signal.

9.3.2.11 VI_ATTR_ASRL_DCD_STATE

Information

RW Global	ViInt16	VI_STATE_ASSERTED, VI_STATE_UNASSERTED, VI_STATE_UNKNOWN

Description

This attribute shows the current state of the Data Carrier Detect (DCD) input signal. The DCD signal is often used by modems to indicate the detection of a carrier (remote modem) on the telephone line. The DCD signal is also known as “Receive Line Signal Detect (RLSD).”

9.3.2.12 VI_ATTR_ASRL_DSR_STATE

Information

RW Global	ViInt16	VI_STATE_ASSERTED, VI_STATE_UNASSERTED, VI_STATE_UNKNOWN

Description

This attribute shows the current state of the Data Set Ready (DSR) input signal.

9.3.2.13 VI_ATTR_ASRL_DTR_STATE

Information

RW Global	ViInt16	VI_STATE_ASSERTED, VI_STATE_UNASSERTED, VI_STATE_UNKNOWN

Description

This attribute is used to manually assert or unassert the Data Terminal Ready (DTR) output signal.

9.3.2.14 VI_ATTR_ASRL_RI_STATE

Information

RW Global	ViInt16	VI_STATE_ASSERTED, VI_STATE_UNASSERTED, VI_STATE_UNKNOWN

Description

This attribute shows the current state of the Ring Indicator (RI) input signal. The RI signal is often used by modems to indicate that the telephone line is ringing.

9.3.2.15 VI_ATTR_ASRL_RTS_STATE

Information

RW Global	ViInt16	VI_STATE_ASSERTED, VI_STATE_UNASSERTED, VI_STATE_UNKNOWN

Description

This attribute is used to manually assert or unassert the Request To Send (RTS) output signal. When the VI_ATTR_ASRL_FLOW_CNTRL attribute is set to VI_ASRL_FLOW_RTS_CTS, this attribute is ignored when changed, but can be read to determine whether the background flow control is asserting or unasserting the signal.

9.3.2.16 VI_ATTR_ASRL_REPLACE_CHAR

Information

RW Local	ViUInt8	0 to FFh

Description

This attribute specifies the character to be used to replace incoming characters that arrive with errors (such as parity error).

9.3.2.17 VI_ATTR_ ASRL_XON_CHAR

Information

RW Local	ViUInt8	0 to FFh

Description

This attribute specifies the value of the XON character used for XON/XOFF flow control (both directions). If XON/XOFF flow control (software handshaking) is not being used, the value of this attribute is ignored.

9.3.2.18 VI_ATTR_ ASRL_XOFF_CHAR

Information

RW Local	ViUInt8	0 to FFh

Description

This attribute specifies the value of the XOFF character used for XON/XOFF flow control (both directions). If XON/XOFF flow control (software handshaking) is not being used, the value of this attribute is ignored.

9.3.2.19 VI_ATTR_DMA_ALLOW_EN

Information

RW Local	ViBoolean	VI_TRUE, VI_FALSE

Description

This attribute specifies whether I/O accesses should use DMA (`VI_TRUE`) or Programmed I/O (`VI_FALSE`).

9.3.2.20 VI_ATTR_FILE_APPEND_EN

Information

RW Local	`ViBoolean`	`VI_TRUE`, `VI_FALSE`

Description

This attribute specifies whether `viReadToFile()` will overwrite (truncate) or append when opening a file.

9.3.2.21 VI_ATTR_GPIB_PRIMARY_ADDR

Information

RO Global	`ViUInt16`	0 to 30

Description

Primary address of the GPIB device used by the given session.

9.3.2.22 VI_ATTR_GPIB_READDR_EN

Information

R/W Local	`ViBoolean`	`VI_TRUE`, `VI_FALSE`

Description

This attribute specifies whether to use repeat addressing before each read or write operation.

9.3.2.23 VI_ATTR_GPIB_REN_STATE

Information

RO Global	`ViInt16`	`VI_STATE_ASSERTED`, `VI_STATE_UNASSERTED`, `VI_STATE_UNKNOWN`

Description

This attribute returns the current state of the GPIB REN interface line.

9.3.2.24 VI_ATTR_GPIB_SECONDARY_ADDR

Information

RO Global	`ViUInt16`	0 to 31, `VI_NO_SEC_ADDR`

Description

Secondary address of the GPIB device used by the given session.

9.3.2.25 VI_ATTR_GPIB_UNADDR_EN

Information

R/W Local	`ViBoolean`	`VI_TRUE`, `VI_FALSE`

Description

This attribute specifies whether to unaddress the device (UNT and UNL) after each read or write operation.

9.3.2.26 VI_ATTR_INTF_INST_NAME

Information

RO Global	`ViString`	N/A

Description

Human-readable text describing the given interface.

9.3.2.27 VI_ATTR_INTF_NUM

Information

RO Global	`ViUInt16`	0 to FFFFh

Description

Board number for the given interface.

9.3.2.28 VI_ATTR_INTF_TYPE

Information

RO Global	`ViUInt16`	`VI_INTF_VXI`, `VI_INTF_GPIB`, `VI_INTF_GPIB_VXI`, `VI_INTF_ASRL`, `VI_INTF_TCPIP`, `VI_INTF_USB`

Description

Interface type of the given session.

9.3.2.29 VI_ATTR_IO_PROT

Information

R/W Local	`ViUInt16`	`VI_PROT_NORMAL`, `VI_PROT_FDC`, `VI_PROT_HS488`, `VI_PROT_4882_STRS`, `VI_PROT_USBTMC_VENDOR`

Description

Specifies which protocol to use. In GPIB, you can choose between normal and high speed (HS488) data transfers. In ASRL and TCPIP systems, you can choose between normal and 488-style transfers, in which case the `viAssertTrigger()` and `viReadSTB()` operations send 488.2-defined strings.

9.3.2.30 VI_ATTR_MANF_ID

Information

RO Global	`ViUInt16`	0 to FFFFh

Description

Manufacturer identification number of the device.

9.3.2.31 VI_ATTR_MANF_NAME

Information

RO Global	`ViString`	N/A

Description

This string attribute is the manufacturer's name. The value of this attribute should be used for display purposes only and not for programmatic decisions, as the value can be different between VISA implementations and/or revisions.

9.3.2.32 VI_ATTR_MODEL_CODE

Information

RO Global	`ViUInt16`	0 to FFFFh

Description

Model code for the device.

9.3.2.33 VI_ATTR_MODEL_NAME

Information

RO Global	`ViString`	N/A

Description

This string attribute is the model name of the device. The value of this attribute should be used for display purposes only and not for programmatic decisions, as the value can be different between VISA implementations and/or revisions.

9.3.2.34 VI_ATTR_RD_BUF_OPER_MODE

Information

R/W Local	`ViUInt16`	`VI_FLUSH_ON_ACCESS`, `VI_FLUSH_DISABLE`

Description

Determines the operational mode of the read buffer. When the operational mode is set to `VI_FLUSH_DISABLE` (default), the buffer is flushed only on explicit calls to `viFlush()`.

If the operational mode is set to `VI_FLUSH_ON_ACCESS`, the buffer is flushed every time a `viScanf()` operation completes.

9.3.2.35 VI_ATTR_RD_BUF_SIZE

Information

RO Local	`ViUInt32`	N/A

Description

This attribute specifies the size of the formatted I/O read buffer. The user can modify this value by calling `viSetBuf()`.

9.3.2.36 VI_ATTR_SEND_END_EN

Information

R/W Local	`ViBoolean`	`VI_TRUE`, `VI_FALSE`

Description

Whether to assert END during the transfer of the last byte of the buffer.

9.3.2.37 VI_ATTR_SUPPRESS_END_EN

Information

R/W Local	`ViBoolean`	`VI_TRUE`, `VI_FALSE`

Description

Whether to suppress the END indicator termination. If this attribute is set to `VI_TRUE`, the END indicator does not terminate read operations. If this attribute is set to `VI_FALSE`, the END indicator terminates read operations.

9.3.2.38 VI_ATTR_TCPIP_ADDR

Information

RO Global	`ViString`	N/A

Description

This is the TCPIP address of the device to which the session is connected. This string is formatted in dot-notation.

9.3.2.39 VI_ATTR_TCPIP_DEVICE_NAME

Information

RO Global	`ViString`	N/A

Description

This specifies the LAN device name used by the VXI-11 or HiSLIP protocol during connection.

9.3.2.40 VI_ATTR_TCPIP_HISLIP_MAX_MESSAGE_KB

Information

R/W Local	`ViUInt32`	0h – ffffffffh

Description

This is the maximum HiSLIP message size VISA will accept from a HiSLIP system in units of kilobytes (1024 bytes). Defaults to 1024 (a 1 MB maximum message size).

9.3.2.41 VI_ATTR_TCPIP_HISLIP_OVERLAP_EN

Information

R/W Local	`ViBoolean`	`VI_TRUE`, `VI_FALSE`

Description

This enables HiSLIP ‘Overlap’ mode and its value defaults to the mode suggested by the instrument on HiSLIP connection. If disabled, the connection uses ‘Synchronous’ mode to detect and recover from interrupted errors. If enabled, the connection uses ‘Overlapped’ mode to allow overlapped responses. If changed, VISA will do a Device Clear operation to change the mode.

9.3.2.42 VI_ATTR_TCPIP_HISLIP_VERSION

Information

RO Local	`ViVersion`	N/A

Description

This is the HiSLIP protocol version used for a particular HiSLIP connetion. Currently, HiSLIP version 1.0 would return a `ViVersion` value of 0x00100000.

9.3.2.43 VI_ATTR_TCPIP_HOSTNAME

Information

RO Global	`ViString`	N/A

Description

This specifies the host name of the device. If no host name is available, this attribute returns an empty string.

9.3.2.44 VI_ATTR_TCPIP_IS_HISLIP

Information

RO Global	`ViBoolean`	`VI_TRUE`, `VI_FALSE`

Description

Specifies whether this resource uses the HiSLIP protocol.

9.3.2.45 VI_ATTR_TCPIP_PORT

Information

RO Global	`ViUInt16`	0 to FFFFh

Description

This specifies the port number for a given TCPIP address. For a TCPIP SOCKET resource, this is a required part of the address string.

9.3.2.46 VI_ATTR_TERMCHAR

Information

R/W Local	ViUInt8	0 to FFh

Description

Termination character. When the termination character is read and VI_ATTR_TERMCHAR_EN is enabled during a read operation, the read operation terminates.

9.3.2.47 VI_ATTR_TERMCHAR_EN

Information

R/W Local	ViBoolean	VI_TRUE, VI_FALSE

Description

Flag that determines whether the read operation should terminate when a termination character is received.

9.3.2.48 VI_ATTR_TMO_VALUE

Information

R/W Local	ViUInt32	VI_TMO_IMMEDIATE,1 to FFFFFFFEh, VI_TMO_INFINITE

Description

Minimum timeout value to use, in milliseconds. A timeout value of VI_TMO_IMMEDIATE means that operations should never wait for the device to respond. A timeout value of VI_TMO_INFINITE disables the timeout mechanism.

9.3.2.49 VI_ATTR_TRIG_ID

Information

R/W Local	ViInt16	VI_TRIG_TTL0 to VI_TRIG_TTL7; VI_TRIG_ECL0 to VI_TRIG_ECL5; VI_TRIG_STAR_SLOT1 to VI_TRIG_STAR_SLOT12; VI_TRIG_STAR_VXI0 to VI_TRIG_STAR_VXI2

Description

Identifier for the current triggering mechanism.

9.3.2.50 VI_ATTR_USB_INTFC_NUM

Information

RO Global	ViInt16	0 to 254

Description

Specifies the USB interface number of this device to which this session is connected

9.3.2.51 VI_ATTR_USB_MAX_INTR_SIZE

Information

RW Local	`ViUInt16`	0 to FFFFh

Description

Specifies the maximum number of bytes that this USB device will send on the interrupt IN pipe. The default value is the same as the maximum packet size of the interrupt IN pipe.

9.3.2.52 VI_ATTR_USB_PROTOCOL

Information

RO Global	`ViInt16`	0 to 255

Description

Specifies the USB protocol number.

9.3.2.53 VI_ATTR_USB_SERIAL_NUM

Information

RO Global	`ViString`	N/A

Description

This string attribute is the serial number of the USB instrument. The value of this attribute should be used for display purposes only and not for programmatic decisions.

9.3.2.54 VI_ATTR_WR_BUF_OPER_MODE

Information

R/W Local	`ViUInt16`	`VI_FLUSH_ON_ACCESS`, `VI_FLUSH_WHEN_FULL`

Description

Determines the operational mode of the write buffer. When the operational mode is set to `VI_FLUSH_WHEN_FULL` (default), the buffer is flushed when an END indicator is written to the buffer, or when the buffer fills up.

If the operational mode is set to `VI_FLUSH_ON_ACCESS`, the write buffer is flushed under the same conditions, and also every time a `viPrintf()` operation completes.

9.3.2.55 VI_ATTR_WR_BUF_SIZE

Information

RO Local	`ViUInt32`	N/A

Description

This attribute specifies the size of the formatted I/O write buffer. The user can modify this value by calling `viSetBuf()`.

9.3.3 Instrument class: INTFC

9.3.3.1 VI_ATTR_DEV_STATUS_BYTE

Information

RW Global	`ViUInt8`	0 to FFh

Description

This attribute specifies the 488-style status byte of the local controller associated with this session.

If this attribute is written and bit 6 (0x40) is set, this device or controller will assert a service request (SRQ) if it is defined for this interface.

9.3.3.2 VI_ATTR_DMA_ALLOW_EN

Information

RW Local	`ViBoolean`	`VI_TRUE`, `VI_FALSE`

Description

This attribute specifies whether I/O accesses should use DMA (`VI_TRUE`) or Programmed I/O (`VI_FALSE`).

9.3.3.3 VI_ATTR_FILE_APPEND_EN

Information

RW Local	`ViBoolean`	`VI_TRUE`, `VI_FALSE`

Description

This attribute specifies whether `viReadToFile()` will overwrite (truncate) or append when opening a file.

9.3.3.4 VI_ATTR_GPIB_ADDR_STATE

Information

RO Global	`ViInt16`	`VI_GPIB_UNADDRESSED`, `VI_GPIB_TALKER`, `VI_GPIB_LISTENER`

Description

This attribute shows whether the specified GPIB interface is currently addressed to talk or listen, or is not addressed.

9.3.3.5 VI_ATTR_GPIB_ATN_STATE

Information

RO Global	`ViInt16`	`VI_STATE_ASSERTED`, `VI_STATE_UNASSERTED`, `VI_STATE_UNKNOWN`

Description

This attribute shows the current state of the GPIB ATN (ATtentioN) interface line.

9.3.3.6 VI_ATTR_GPIB_CIC_STATE

Information

RO Global	ViBoolean	VI_TRUE, VI_FALSE	

Description

This attribute shows whether the specified GPIB interface is currently CIC (controller in charge).

9.3.3.7 VI_ATTR_GPIB_HS488_CBL_LEN

Information

RW Global	ViInt16	1 to 15, VI_GPIB_HS488_NIMPL	VI_GPIB_HS488_DISABLED,

Description

This attribute specifies the total number of meters of GPIB cable used in the specified GPIB interface. If HS488 is not implemented, querying this attribute should return the value VI_GPIB_HS488_NIMPL. On these systems, trying to set this attribute value will return the error VI_ERROR_NSUP_ATTR_STATE.

9.3.3.8 VI_ATTR_GPIB_NDAC_STATE

Information

RO Global	ViInt16	VI_STATE_ASSERTED, VI_STATE_UNKNOWN	VI_STATE_UNASSERTED,

Description

This attribute shows the current state of the GPIB NDAC (Not Data ACcepted) interface line.

9.3.3.9 VI_ATTR_GPIB_PRIMARY_ADDR

Information

RW Global	ViUInt16	0 to 30

Description

Primary address of the local GPIB controller used by the given session.

9.3.3.10 VI_ATTR_GPIB_REN_STATE

Information

RO Global	ViInt16	VI_STATE_ASSERTED, VI_STATE_UNKNOWN	VI_STATE_UNASSERTED,

Description

This attribute returns the current state of the GPIB REN (Remote ENable) interface line.

9.3.3.11 VI_ATTR_GPIB_SECONDARY_ADDR

Information

RW Global	ViUInt16	0 to 31, VI_NO_SEC_ADDR

Description

Secondary address of the local GPIB controller used by the given session.

9.3.3.12 VI_ATTR_GPIB_SRQ_STATE

Information

RO Global	ViInt16	VI_STATE_ASSERTED, VI_STATE_UNASSERTED, VI_STATE_UNKNOWN

Description

This attribute shows the current state of the GPIB SRQ (Service ReQuest) interface line.

9.3.3.13 VI_ATTR_GPIB_SYS_CNTRL_STATE

Information

RW Global	ViBoolean	VI_TRUE, VI_FALSE

Description

This attribute shows whether the specified GPIB interface is currently the system controller. In some implementations, this attribute may be modified only through a configuration utility. On these systems, this attribute is read only (RO).

9.3.3.14 VI_ATTR_INTF_INST_NAME

Information

RO Global	ViString	N/A

Description

Human-readable text describing the given interface.

9.3.3.15 VI_ATTR_INTF_NUM

Information

RO Global	ViUInt16	0 to FFFFh

Description

Board number for the given interface.

9.3.3.16 VI_ATTR_INTF_TYPE

Information

RO Global	ViUInt16	VI_INTF_VXI, VI_INTF_GPIB, VI_INTF_GPIB_VXI, VI_INTF_TCPIP, VI_INTF_USB

Description

Interface type of the given session.

9.3.3.17 VI_ATTR_RD_BUF_OPER_MODE

Information

R/W Local	ViUInt16	VI_FLUSH_ON_ACCESS, VI_FLUSH_DISABLE

Description

Determines the operational mode of the read buffer. When the operational mode is set to VI_FLUSH_DISABLE (default), the buffer is flushed only on explicit calls to viFlush().

If the operational mode is set to VI_FLUSH_ON_ACCESS, the buffer is flushed every time a viScanf() operation completes.

9.3.3.18 VI_ATTR_RD_BUF_SIZE

Information

RO Local	ViUInt32	N/A

Description

This attribute specifies the size of the formatted I/O read buffer. The user can modify this value by calling viSetBuf().

9.3.3.19 VI_ATTR_SEND_END_EN

Information

R/W Local	ViBoolean	VI_TRUE, VI_FALSE

Description

Whether to suppress the END indicator termination. If this attribute is set to VI_TRUE, the END indicator does not terminate read operations. If this attribute is set to VI_FALSE, the END indicator terminates read operations.

9.3.3.20 VI_ATTR_TERMCHAR

Information

R/W Local	ViUInt8	0 to FFh

Description

Termination character. When the termination character is read and VI_ATTR_TERMCHAR_EN is enabled during a read operation, the read operation terminates.

9.3.3.21 VI_ATTR_TERMCHAR_EN

Information

R/W Local	ViBoolean	VI_TRUE, VI_FALSE

Description

Flag that determines whether the read operation should terminate when a termination character is received.

9.3.3.22 VI_ATTR_TMO_VALUE

Information

R/W Local	ViUInt32	VI_TMO_IMMEDIATE,1 to FFFFFFFEh, VI_TMO_INFINITE

Description

Minimum timeout value to use, in milliseconds. A timeout value of VI_TMO_IMMEDIATE means that operations should never wait for the device to respond. A timeout value of VI_TMO_INFINITE disables the timeout mechanism.

9.3.3.23 VI_ATTR_WR_BUF_OPER_MODE

Information

R/W Local	ViUInt16	VI_FLUSH_ON_ACCESS, VI_FLUSH_WHEN_FULL

Description

Determines the operational mode of the write buffer. When the operational mode is set to VI_FLUSH_WHEN_FULL (default), the buffer is flushed when an END indicator is written to the buffer, or when the buffer fills up.

If the operational mode is set to VI_FLUSH_ON_ACCESS, the write buffer is flushed under the same conditions, and also every time a viPrintf() operation completes.

9.3.3.24 VI_ATTR_WR_BUF_SIZE

Information

RO Local	ViUInt32	N/A

Description

This attribute specifies the size of the formatted I/O write buffer. The user can modify this value by calling viSetBuf().

9.3.4 Instrument class: SOCKET

9.3.4.1 VI_ATTR_DMA_ALLOW_EN

Information

RW Local	ViBoolean	VI_TRUE, VI_FALSE

Description

This attribute specifies whether I/O accesses should use DMA (VI_TRUE) or Programmed I/O (VI_FALSE).

9.3.4.2 VI_ATTR_FILE_APPEND_EN

Information

RW Local	`ViBoolean`	`VI_TRUE`, `VI_FALSE`

Description

This attribute specifies whether `viReadToFile()` will overwrite (truncate) or append when opening a file.

9.3.4.3 VI_ATTR_INTF_INST_NAME

Information

RO Global	`ViString`	N/A

Description

Human-readable text describing the given interface.

9.3.4.4 VI_ATTR_INTF_NUM

Information

RO Global	`ViUInt16`	0 to FFFFh

Description

Board number for the given interface.

9.3.4.5 VI_ATTR_INTF_TYPE

Information

RO Global	`ViUInt16`	`VI_INTF_VXI`, `VI_INTF_GPIB`, `VI_INTF_GPIB_VXI`, `VI_INTF_TCPIP`, `VI_INTF_USB`

Description

Interface type of the given session.

9.3.4.6 VI_ATTR_IO_PORT

Information

R/W Local	`ViUInt16`	`VI_PROT_NORMAL`, `VI_PROT_FDC`, `VI_PROT_HS488`, `VI_PROT_4882_STRS`, `VI_PROT_USBTMC_VENDOR`

Description

Specifies which protocol to use. In GPIB, you can choose between normal and high speed (HS488) data transfers. In TCPIP systems, you can choose between normal and 488-style transfers, in which case the `viAssertTrigger()` and `viReadSTB()` operations send 488.2-defined strings.

9.3.4.7 VI_ATTR_RD_BUF_OPER_MODE

Information

R/W Local	ViUInt16	VI_FLUSH_ON_ACCESS, VI_FLUSH_DISABLE

Description

Determines the operational mode of the read buffer. When the operational mode is set to VI_FLUSH_DISABLE (default), the buffer is flushed only on explicit calls to viFlush().

If the operational mode is set to VI_FLUSH_ON_ACCESS, the buffer is flushed every time a viScanf() operation completes.

9.3.4.8 VI_ATTR_RD_BUF_SIZE

Information

RO Local	ViUInt32	N/A

Description

This attribute specifies the size of the formatted I/O read buffer. The user can modify this value by calling viSetBuf().

9.3.4.9 VI_ATTR_SEND_END_EN

Information

R/W Local	ViBoolean	VI_TRUE, VI_FALSE

Description

Whether to suppress the END indicator termination. If this attribute is set to VI_TRUE, the END indicator does not terminate read operations. If this attribute is set to VI_FALSE, the END indicator terminates read operations.

9.3.4.10 VI_ATTR_TCPIP_ADDR

Information

RO Global	ViString	N/A

Description

This is the TCPIP address of the device to which the session is connected. This string is formatted in dot-notation.

9.3.4.11 VI_ATTR_TCPIP_HOSTNAME

Information

RO Global	ViString	N/A

Description

This specifies the host name of the device. If no host name is available, this attribute returns an empty string.

9.3.4.12 VI_ATTR_TCPIP_KEEPALIVE

Information

R/W Local	ViBoolean	VI_TRUE, VI_FALSE

Description

An application can request that a TCP/IP provider enable the use of “keep-alive” packets on TCP connections by turning on this attribute. If a connection is dropped as a result of “keep-alives,” the error code VI_ERROR_CONN_LOST is returned to current and subsequent I/O calls on the session.

9.3.4.13 VI_ATTR_TCPIP_NODELAY

Information

R/W Local	ViBoolean	VI_TRUE, VI_FALSE

Description

The Nagle algorithm is disabled when this attribute is enabled (and vice versa). The Nagle algorithm improves network performance by buffering “send” data until a full-size packet can be sent. This attribute is enabled by default in VISA to verify that synchronous writes get flushed immediately.

9.3.4.14 VI_ATTR_TCPIP_PROT

Information

RO Global	ViUInt16	0 to FFFFh

Description

This specifies the port number for a given TCPIP address. For a TCPIP SOCKET resource, this is a required part of the address string.

9.3.4.15 VI_ATTR_TERMCHAR

Information

R/W Local	ViUInt8	0 to FFh

Description

Termination character. When the termination character is read and VI_ATTR_TERMCHAR_EN is enabled during a read operation, the read operation terminates.

9.3.4.16 VI_ATTR_TERMCHAR_EN

Information

R/W Local	ViBoolean	VI_TRUE, VI_FALSE

Description

Flag that determines whether the read operation should terminate when a termination character is received.

9.3.4.17 VI_ATTR_TMO_VALUE

Information

R/W Local	ViUInt32	VI_TMO_IMMEDIATE,1 to FFFFFFFEh, VI_TMO_INFINITE	

Description

Minimum timeout value to use, in milliseconds. A timeout value of VI_TMO_IMMEDIATE means that operations should never wait for the device to respond. A timeout value of VI_TMO_INFINITE disables the timeout mechanism.

9.3.4.18 VI_ATTR_WR_BUF_OPER_MODE

Information

R/W Local	ViUInt16	VI_FLUSH_ON_ACCESS, VI_FLUSH_WHEN_FULL

Description

Determines the operational mode of the write buffer. When the operational mode is set to VI_FLUSH_WHEN_FULL (default), the buffer is flushed when an END indicator is written to the buffer, or when the buffer fills up.

If the operational mode is set to VI_FLUSH_ON_ACCESS, the write buffer is flushed under the same conditions, and also every time a viPrintf() operation completes.

9.3.4.19 VI_ATTR_WR_BUF_SIZE

Information

RO Local	ViUInt32	N/A

Description

This attribute specifies the size of the formatted I/O write buffer. The user can modify this value by calling viSetBuf().

9.4 Events

In the following sections VISA events are presented.

9.4.1 VI_EVENT_SERVICE_REQ

Notification that a service request was received from the device.

Available attributes

- VI_ATTR_EVENT_TYPE: Unique logical identifier of the event. (ViEventType)

10 Index