LabVIEW VISA Tutorial

Introduction

This tutorial gives an overview of LabVIEW's implementation of the VISA language. It also teaches the basic concepts involved in programming instruments with VISA and gives examples demonstrating simple VISA concepts.

Covered Material

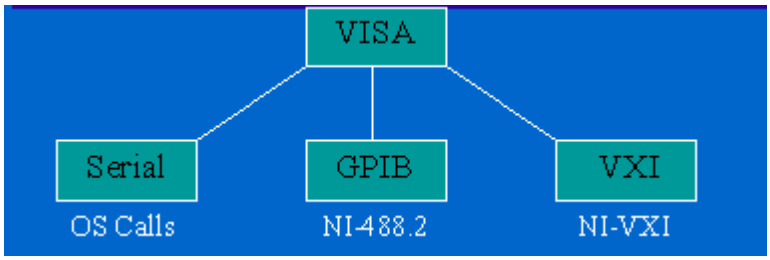
A. VISA Overview
   1. VISA Structure
   2. Supported Platforms and Environments

B. VISA Programming
1. Default Resource Manager, Sessions, Instrument Descriptors.
2. Message-Based Communication
3. Register-Based Communication
4. VISA Attributes

VISA Overview

VISA Structure

VISA is a standard I/O language for instrumentation programming. VISA by itself does not provide instrumentation programming capability. VISA is a high-level API that calls into lower level drivers. The hierarchy of NI-VISA is shown in the figure below.



VISA is capable of controlling VXI, GPIB, or Serial instruments and makes the appropriate driver calls depending on the type of instrument being used. When debugging VISA problems it is important to keep in mind that this hierarchy exists. An apparent VISA problem could in reality be the results of a bug or installation problem with one of the drivers into which VISA is calling.

VISA Advantages

One of VISA's advantages is that it uses many of the same operations to communicate with instruments regardless of the interface type. For example, the VISA command to write an ASCII string to a message-based instrument is the same whether the instrument is Serial, GPIB, or VXI. Thus, VISA provides interface independence. This can make it easy to switch interfaces and also gives users who must program instruments for different interfaces a single language they can learn.

VISA is also designed so that programs written using VISA function calls are easily portable from one platform to another. To ensure this, VISA strictly defines its own data types such that issues like the size, of an integer variable from one platform to another should not affect a VISA program. The VISA function calls and their associated parameters are uniform across all platforms; software can be ported to other platforms and then recompiled. In other words, a C program using VISA can be ported to other platforms supporting C. A LabVIEW program can be ported to other platforms supporting LabVIEW. At first, the abundance of new VISA data types might seem like a burden, but LabVIEW makes them virtually transparent when using VISA.

Another advantage of VISA is that it is an object-oriented language which will easily adapt to new instrumentation interfaces as they are developed in the future. The interface independence that is built into the language will also make it even easier for programmers to move to new interfaces with VISA in the future.

VISA's greatest advantage, perhaps, is that it is an easy language to learn and use. Its object-oriented structure makes the language and its operations intuitive to learn. This is due in part to the fact that VISA provides interface independence by using the same operations for different interfaces and also by the fact VISA presents a very simple and easy-to-use API. VISA provides the most frequently used functionality for instrumentation programming in a very compact command set.

Supported Platforms And Environments

The platforms and environments that are supported by NI-VISA are shown in the table below.

Platform	Environments
Macintosh	LabVIEW, C
Windows 3.1	MSVC, Borland C++, CVI, LabVIEW, VB
Windows 95/NT	C, CVI, LabVIEW, VB
Solaris 1	CVI, LabVIEW
Solaris 2	CVI, LabVIEW
Hp-Ux	CC, CVI, LabVIEW

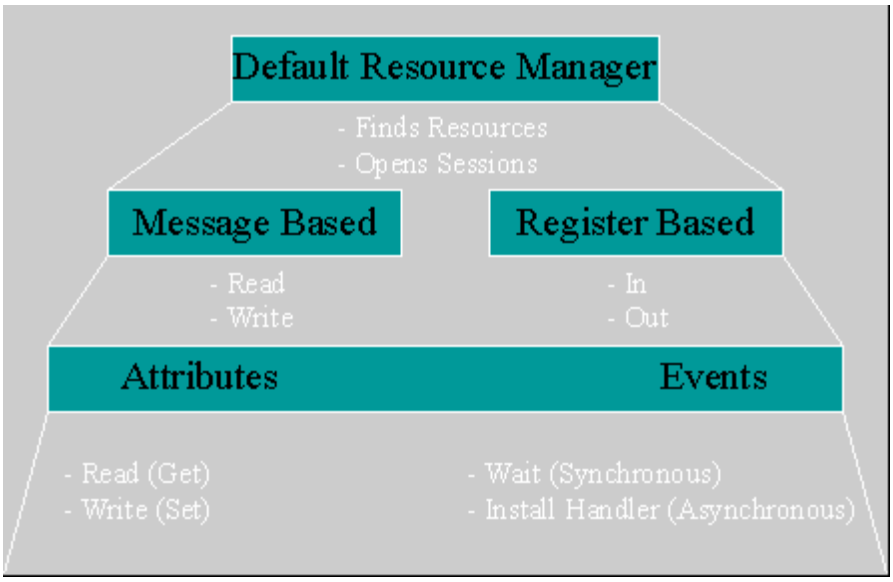
Because VISA is the language used in writing instrument drivers, most instrument drivers currently written by National Instruments supports all of these environments.

VISA Programming

Before getting started with an introduction to VISA programming some terminology needs to be mentioned. These terms will become clearer as VISA's functionality is described.

VISA is an object-oriented language. The most important objects in the VISA language are known as resources. In object-oriented terminology, the functions that can be used with an object are known as operations. In addition to the operations that can be used with an object, the object has variables associated with it that contain information related to the object. In VISA, these variables are known as attributes.

A simplified outline of the internal structure of the VISA language is shown in the diagram below.



There is a default resource manager at the top of the VISA hierarchy that can search for available resources or open sessions to them. Resources can be GPIB, serial, message-based VXI, or register-based VXI. The most common operations for message based instruments are reads and writes. The most common operations for register based instruments are In's and Out's. In addition, resources have a variety of properties associated with them known as attributes. These can be read or modified with Attribute Nodes.

The remaining sections of this lesson will look at these parts of the VISA language in detail and present simple programming examples and exercises performing common programming tasks with the VISA language. The last part of the internal VISA structure shown in the diagram are Events. Events are more advanced topics and are not discussed in this tutorial.

Default Resource Manager, Session, and Instrument Descriptors

The Default Resource Manager is at the highest level of VISA operations. Communication must be established with the Resource Manager at the beginning of any VISA program. This immediately brings up two terms that need to be defined: resource and session.

Resource - An instrument or controller.

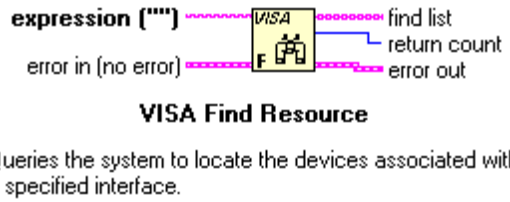
Session - A connection, or link, to the VISA Default Resource Manager or a resource.

The reason the VISA Default Resource Manager is so important is because one of its operations is to open sessions to other resources. Sessions must be opened to the instruments the application will communicate with. Therefore communication with the Default Resource Manager must be established first within an application.

The VISA Default Resource Manager also has another operation that it can carry out. That operation is to search for available resources in the system. Sessions can then be opened to any of these resources.

NOTE: *In LabVIEW, a session to the VISA Default Resource Manager is opened automatically the first time either of the Default Resource Manager operations are used in the program. Therefore, the first step in a LabVIEW VISA application is to open sessions to resources or to search for available resources.*

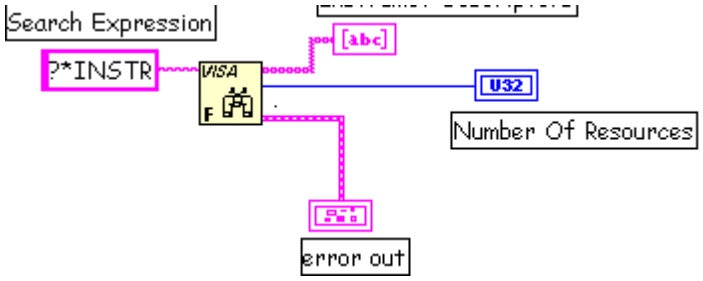
The VISA Find Resources VI that carries out the operation of searching for available resources in the system is shown below. This VI is a common starting point for a VISA program. It can be used to determine if all of the necessary resources for the application to run are available.



The only necessary input to the VISA Find Resources VI is a string called the search expression. This determines what types of resources the Find Resources VI will return. Possible strings for the search expression are shown in the table below and can be found in the LabVIEW Online Reference.

Interface	Expression
GPIB	GPIB[0-9]*::?*INSTR
GPIB-VXI	GPIB-VXI?*INSTR
GPIB or GPIB-VXI	GPIB?*INSTR
VXI	VXI?*INSTR
All VXI	?*VXI[0-9]*::?*INSTR
Serial	ASRL[0-9]*::?*INSTR
All	?*INSTR

The important return values of the VI are the return count (which simply reports the number of resources that were found) and the find list. The find list is an array of strings. Each string contains the description of one of the resources that was found. These strings are known as instrument descriptors. A simple VI that will find all of the available resources in the system is shown in the figure below.



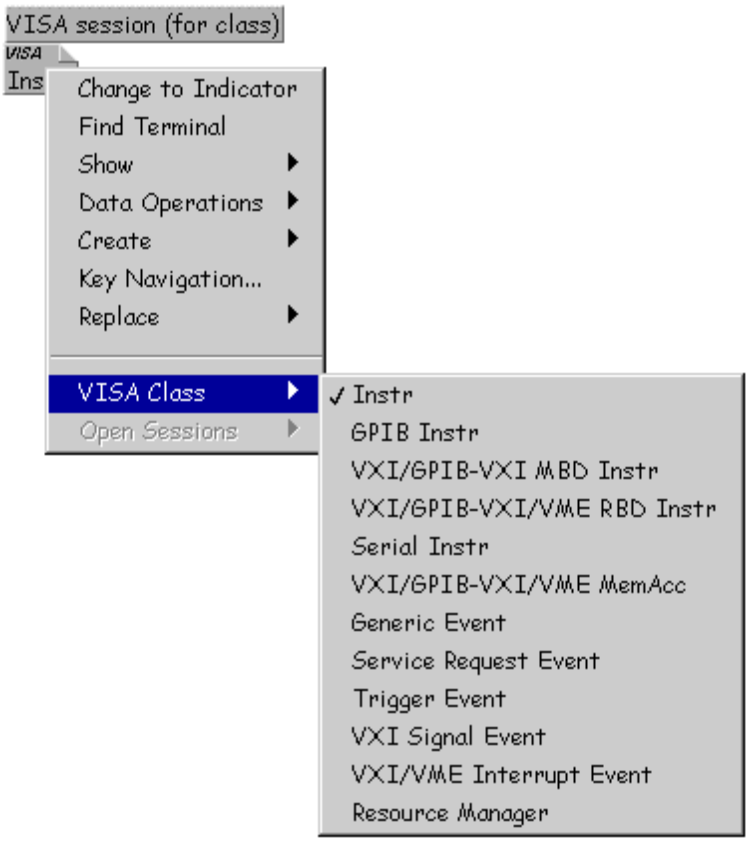
Instrument Descriptor - The exact name and location of a VISA resource. This string has the format

Interface Type[Board Index]::Address::VISA Class

The instrument descriptors are simply specific instruments found by the search query. The board index only needs to be used if more than one interface type is present in the system. For example, if the system contains two GPIB plug-in boards one could be referred to as GPIB0 and one as GPIB1. In this case, the board index needs to be used in instrument descriptors. For VXI instruments, the "Address" parameter is the Logical Address of the instrument. For GPIB instruments, it is the GPIB primary address. For Serial instruments the address is not used. An example is "ASRL1::INSTR" as the descriptor for the COM 1 serial port on a personal computer.

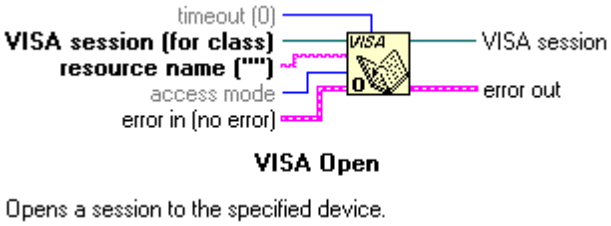
The VISA Class is a grouping that encapsulates some or all of the VISA operations. INSTR is the most general class that encompasses all VISA operations. In the future, other classes may be added to the VISA specification. Currently the VISA Class does not need to be included as part of the instrument descriptor. Nevertheless, this should be done to ensure future compatibility. Currently, most applications simply use the INSTR class.

LabVIEW supplies another way to set the class for a VISA session that can be used now. This is done by popping up on the front panel VISA Session control and selecting the VISA Class as shown in the figure below.



If a class other than the default Instr class is selected, only VIs for operations associated with that device class can be wired successfully with this session. For example, if GPIB Instr is selected for the VISA class, VIs for VXI register accesses can not be wired with the session.

The instrument descriptors are used to open sessions to the resources in the system using the Default Resource Managers ability to open sessions. The VISA Open VI that carries out this operation is shown below.

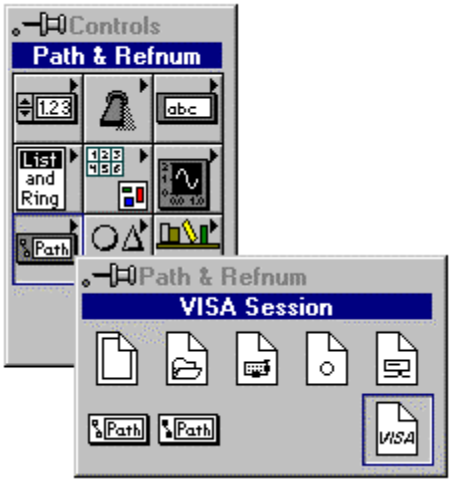


The resource name input is the VISA instrument descriptor for the resource to which a session will be opened.

Note: *The Find Resources VI does not need to be used to obtain instrument descriptors for resources. The VISA Open VI can be used with an instrument descriptor provided by the user or programmer. However, to be sure of the syntax for the descriptor, it is best to run Find Resources first.*

Note: *In most applications, sessions only need to opened once for each instrument that the application will communicate with. This session can be routed throughout the application and then closed at the end of the application.*

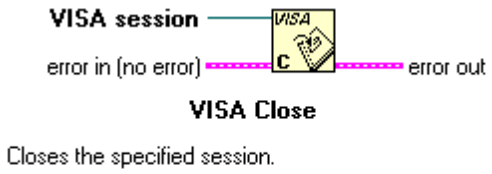
Notice that there is also a VISA session input to the VISA Open VI. In order to open sessions to resources in LabVEIW, a VISA Session front panel control is needed. The VISA session front panel control can be found in the Controls Palette in the Path & Refnum sub-palette.



At this point it is important to have a clear understanding of the Default Resource Manager, instrument descriptors, and sessions. A good analogy can be made between the VISA Default Resource Manager and a telephone operator. Opening a session to the Default Resource Manager (remember this is done automatically in LabVIEW) is like picking up the phone and calling the operator to establish a line of communication between a program and the VISA driver.

In turn, the telephone operator has the ability to dial phone numbers to establish lines of communication with resources in the system. The phone numbers that the Resource Manager uses are the instrument descriptors. The lines of communication are the sessions that are opened to VISA resources. In addition, the Resource Manager can look for all the available phone numbers. This is the VISA Find Resources operation.

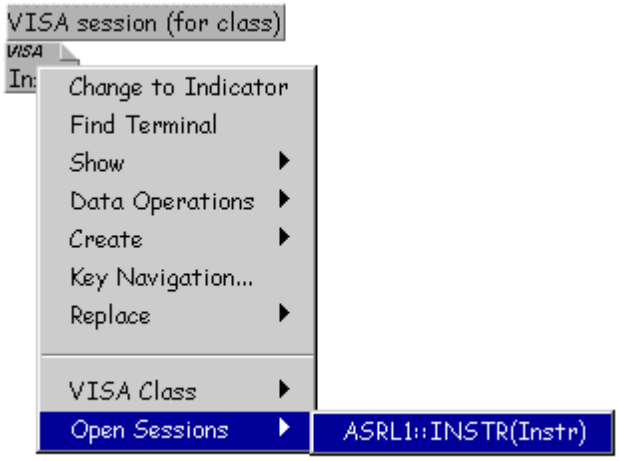
An open session to a VISA resource also uses system resources within the computer. To properly end a VISA program all of the sessions opened to VISA resources should be closed. To do this, there is a VISA Close VI that is shown below.



The important input to the VISA Close VI is the session to be closed. This session originally comes from the output session terminal of the VISA Open VI.

If a session is not closed when a VI is run that session will remain open. It is a good idea to always close any sessions that are opened in an application so open sessions don't build up and cause problems with system resources. However, there are cases when leaving sessions open can be useful.

If a VI runs and leaves a session open without closing it - this session can still be used in later VI's. An already open session can be accessed by popping up on a VISA Session front panel control and going to the Open Sessions selection. The output of the VISA Session front panel control will then be the already open session that is selected. In this way sessions can be closed that were left open from earlier runs of a VI. This method can also be used to interactively test parts of an application. An example of selecting an already open session is shown in the figure below.



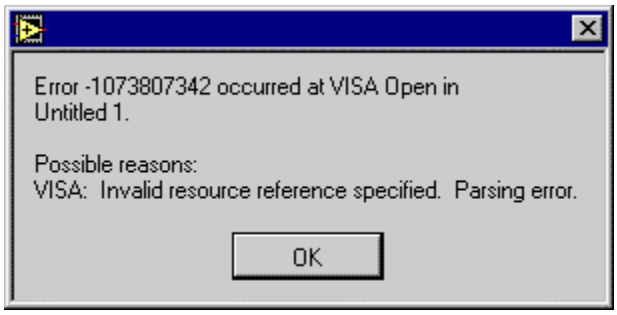
The open session's choice of the front panel VISA Session control can be used to check if any sessions are currently open. Accessing open sessions by popping up on front panel VISA Session Controls also provides a convenient way to interactively run parts of an instrument driver.

Error Handling with VISA

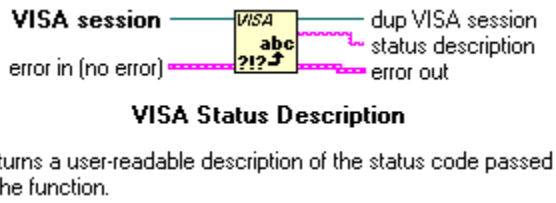
Error handling with VISA VIs is similar to error handling with other I/O VIs in LabVIEW. Each of the VISA VIs contain Error Input and Error Output terminals that are used to pass error clusters from one VI to another in a diagram. The error cluster contains a Boolean flag indicating whether an error has occurred, a numeric VISA error code, and a string containing the location of the VI where the error occurred. If an error occurs, subsequent VIs will not try to execute and will simply pass on the error cluster. A front panel error cluster indicator showing the output from the error out terminal of a VISA function is shown in the figure below.

Notice that in this case an error has occurred because the Boolean value in the error cluster is turned on. Also notice that the VISA error code for the error that has occurred is cut off in the code indicator. VISA error codes are 32-bit integers that are usually referred to in hexadecimal format. The LabVIEW error cluster displays the code in decimal. There is a VISA Error Codes section in the LabVIEW Online Help that also lists all of the error codes in decimal. However, as the figure above shows, these error codes are cut off in the error cluster. The code indicator must be resized to display the entire error code.

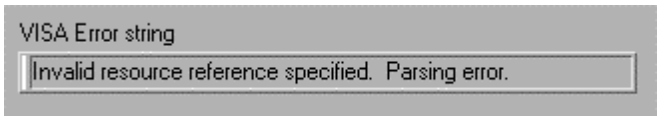
The LabVIEW Simple and General Error Handler VI's can be found in the Time & Dialog sub-palette under the functions palette. These VI's provide a pop-up dialogue box if an error occurs and also look up the error code to determine possible reasons for the error. The result of running the same code that produced the error shown in the error cluster above using the Simple Error Handler VI instead is shown in the figure below.



Notice that the code description is listed under the possible reasons. It is not always convenient to handle errors with pop-up dialogue boxes through the LabVIEW error handling VIs. VISA also provides an operation that will take a VISA error code and produce the error message string corresponding to the code as an output. This VI is shown in the figure below.



The inputs to this VI are a VISA session and a VISA error cluster. The VI will check the VISA code in the error cluster that was passed to it and output the text description of the code from the status description terminal. Even if the input error cluster did have the error value set, a status string description indicating success or a warning will be produced. This VI allows a text description of the status after a VISA VI executes to be obtained and used in a program. The figure below shows a Labview string indicator displaying the error string returned from the VISA Status Description VI.



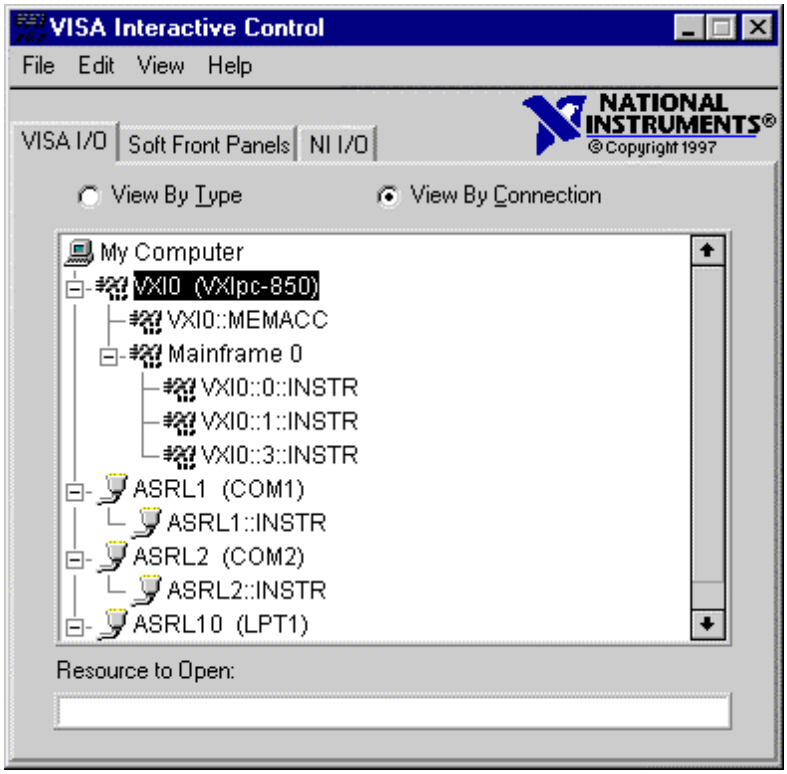
The exact method used for implementing error handling will depend on the nature of the program. However, some sort of error handling mechanism should be implemented in any program involving VISA.

VISAIC And Message-Based Combination

VISAIC

Visa comes with a utility called VISA Interactive Control (VISAIC) on all platforms that support VISA and LabVIEW (except the Macintosh). This utility provides access to all of VISA's functionality interactively, in an easy-to-use graphical environment. It is a convenient starting point for program development and learning about VISA. However, when VISA is installed through LabVIEW's default installation, VISAIC is not installed. VISA can be reinstalled by itself to install this useful utility. The latest versions of VISA are available for no charge on National Instruments ftp site ftp.natinst.com under the support/visa directory.

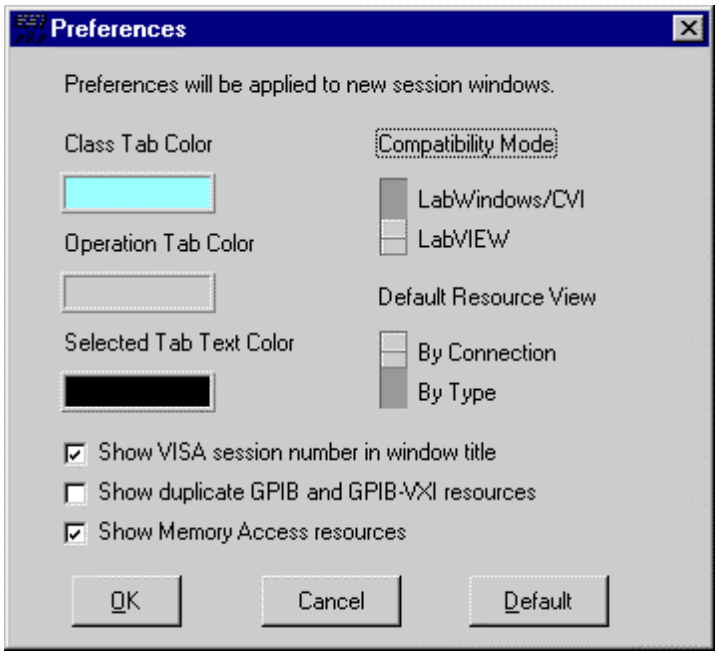
When VISAIC runs, it automatically finds all of the available resources in the system and lists the instrument descriptors for each of these resources under the appropriate resource type. The VISAIC opening window is shown in the figure below.



The Soft Front Panels tab of the main VISAIC panel will give you the option to launch the soft front panels of any VXI plug and play instrument drivers that have been installed on the system.

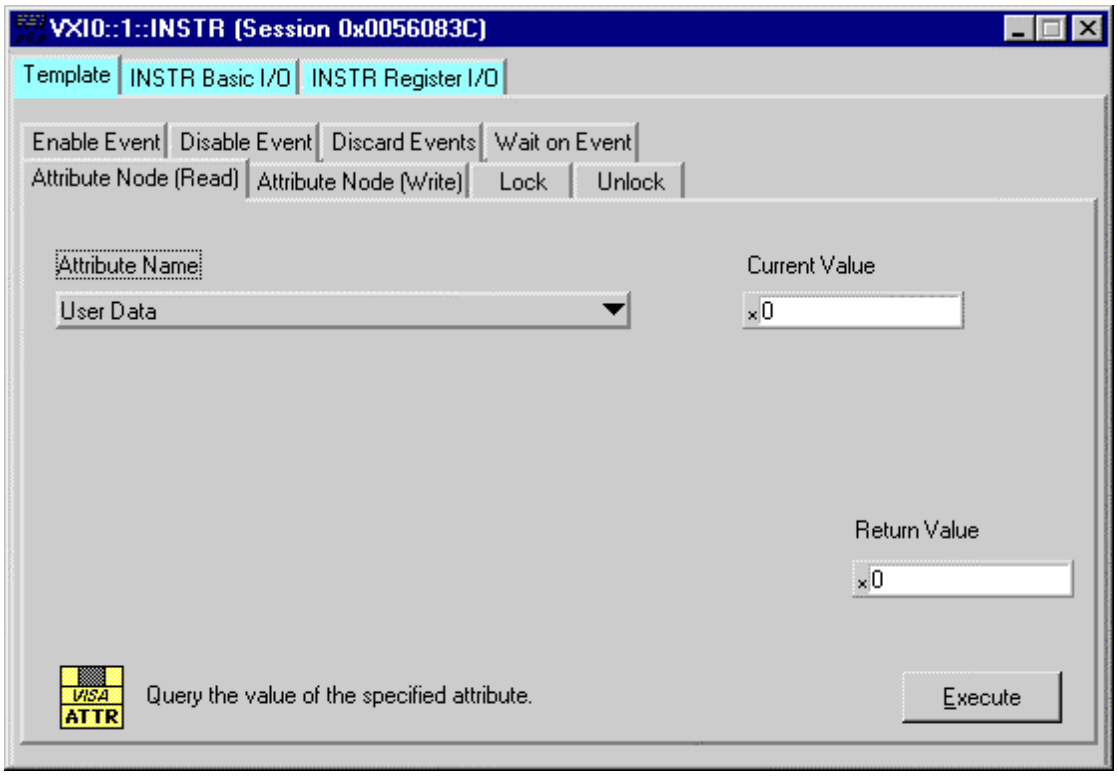
The NI I/O tab will give you the option to launch the NI-VXI interactive utility or the NI-488 interactive utility. This gives you convenient links into the interactive utilities for the drivers VISA calls in case you would like to try debugging at this level.

Double-clicking on any of the instrument descriptors shown in the VISAIC window will open a session to that instrument. Opening a session to the instrument produces a window with a series of tabs for interactively running VISA commands. The exact appearance of these tabs depends on which compatibility mode VISAIC is in. To access the compatibility mode and other VISAIC preferences select Edit >> Preferences . . . to bring up the window shown below.



The VISA implementations are slightly different in LabVIEW and LabWindows/CVI -- these differences are reflected in the operation tabs that are shown when you open a session to a resource. By default, the compatibility mode is set to Labwindows/CVI. This should be changed to LabVIEW if program development is being done with the LabVIEW package. Once the preferences are changed, the new preferences will take effect for any session that is opened later.

When a session to a resource is opened interactively, a window similar to the one shown below will appear.



There are three main tabs that appear in the window. The initial tab is the template tab that contains all of the operations dealing with events, attributes, and locks. Notice that there is a different tab for each of these operations under the main tab. The other main tabs are the INSTR Basic I/O and INSTR Register I/O. The Basic I/O tab contains the basic operations for message-based instruments while the Register I/O tab contains the basic operations for register-based instruments. The Register I/O tab only appears for VXI instruments.

There will be more discussion of the VISA operations in the upcoming sections. For now, it is important to remember that VISAIC performs a VISA Find Resources when it is run and displays instrument descriptors for all of the resources in the system. For VXI instruments, VISAIC also attempts to access one of the configuration registers on each instrument to verify its existence. In upcoming lessons VISAIC will be used as a learning and development tool. It is often easier to develop an application interactively before doing the actual programming. VISAIC can also serve as a very useful debugging tool when problems arise in writing a program or in using a pre-existent application using VISA VI's.

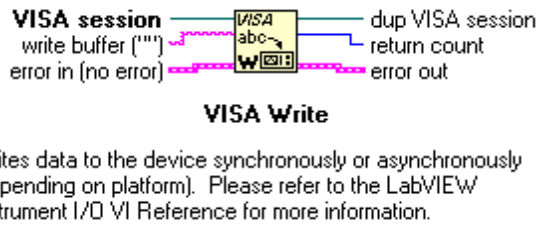
Message-Based Communication

Serial, GPIB, and many VXI devices recognize a variety of message-based command strings. At the VISA level, the actual protocol used to send a command string to an instrument is transparent. A user only needs to know that they would like to write a

message or read a message from a message-based device. The VI's that are used to perform these operations are VISA Write and VISA Read.

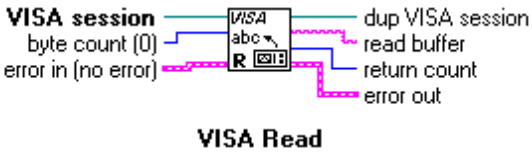
Note: *these same VI's are used to write message based commands to GPIB, Serial, and message-based VXI instrument. VISA knows automatically which functions to call based on the type of resource being used.*

The Write VI is shown below.



The only input, besides the instrument session, is the string to be sent to the instrument.

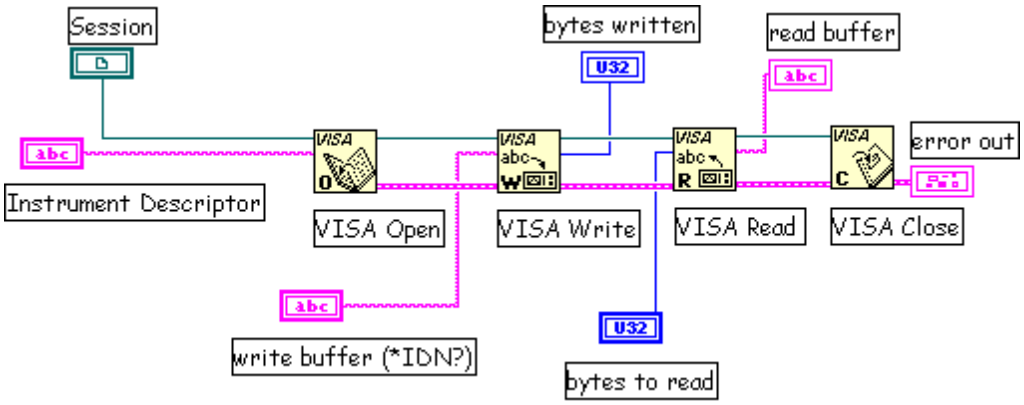
The VISA Read VI is equally easy to use. It is shown below.



The VISA Read VI must be given a byte count input equal to the maximum number of bytes that should be read from the instrument. The VI will terminate the read when this number of bytes has been read or when the end of the transfer is indicated.

The actual message-based commands that the instrument recognizes vary from manufacturer to manufacturer. The GPIB IEEE 488.2 protocol and SCPI standards made some attempts at standardizing the commands for message-based instruments. Many instruments follow these standards. However, the only way to be certain of the commands that should be used with a particular instrument is to refer to the documentation provided by the manufacturer. However, pre-written instrument drivers exist for many message-based devices. These instrument drivers contain functions that put together the appropriate ASCII command strings and send them to the instrument.

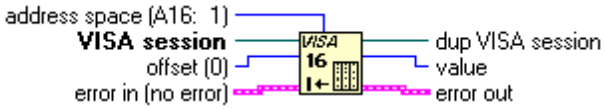
A simple example that writes the *IDN? string to a message-based instrument and reads the response is shown in the figure below.



This program could be used successfully with any device that recognizes the *IDN? command. The device could be Serial, GPIB, or VXI. The only change would be the instrument descriptor.

Register-Based Communication

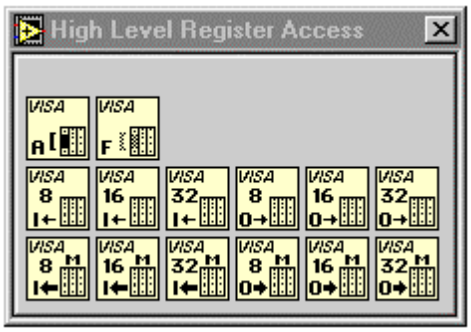
VISA contains a set of register access VIs for use with VXI instruments. Some VXI instruments do not support message-based commands. The only way to communicate with these instruments is through register accesses. All VXI instruments have configuration registers in the upper 16 kilobytes of A16 memory space. Therefore, register access functions can also be used to read from and write to the configuration registers for message-based devices. The basic VISA operation used to read a value from a register is VISA In. There are actually three different versions of this operation for reading a 8, 16, or 32 bit value. The VISA In 16 VI is shown in the figure below.



VISA In 16

Reads in a 16-bit word from the specified address space (assigned memory base + offset). Valid address space values are:
A16 Space: 1
A24 Space: 2
A32 Space: 3

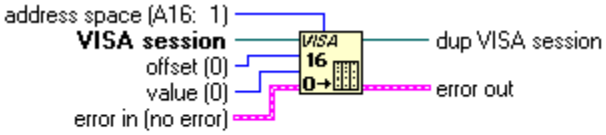
This VI and other basic register access VI's can be found in the High Level Register Access sub-palette under the main VISA function palette.



The address space input simply indicates which VXI address space is to be used. The offset input sometimes causes confusion. Remember that VISA keeps track of the base memory address that a device requests in each address space. The offset input is **relative** to this base address.

Consider the following example. Suppose that you have a device at logical address 1 and would like to use the VISA In 16 VI to read its ID/Logical Address configuration register. You know that this register is at absolute address 0xC040 in A16 space and that the configuration registers for the device at logical address 1 range from 0xC040 to 0xC080. However, VISA also knows this and you only need to specify the offset in the region you wish to access. In this case, that offset is zero.

There is another set of high level register access operations that parallel the VISA In operations, but are for writing to registers. These operations are the VISA Out operations. The VISA Out 16 VI is shown below.

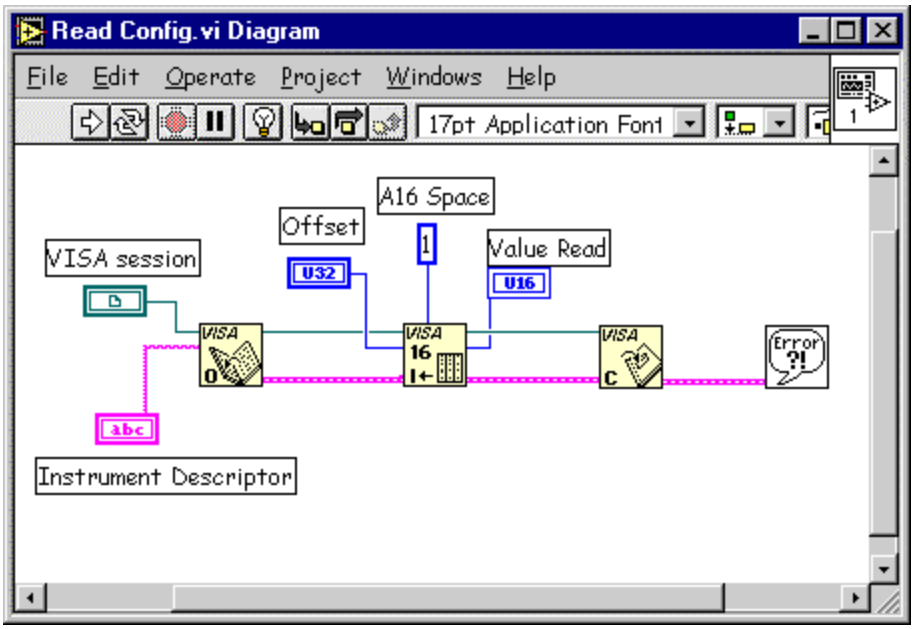


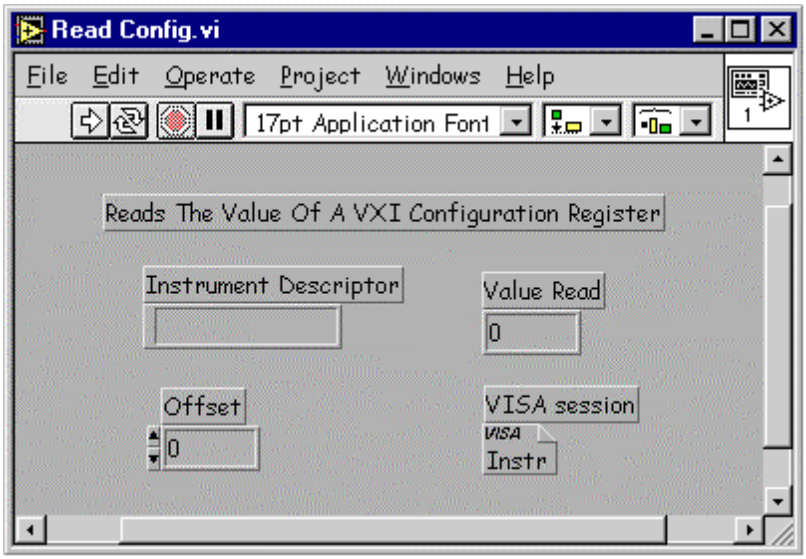
VISA Out 16

Writes a 16-bit word to the specified address space (assigned memory base + offset). Valid address space values are:
A16 Space: 1
A24 Space: 2
A32 Space: 3

This VI is similar to the VISA In 16 VI except the value to write must be provided to the value terminal. Keep in mind when using the VISA Out VIs that some registers can not be accessed.

An example of using the high level VISA access functions in a VI is shown in the simple program below.





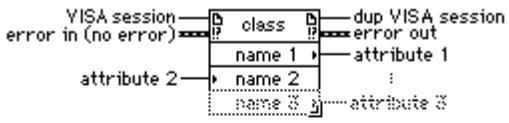
The program asks the user for the instrument descriptor and the offset of the configuration register to be read and returns the value from the register. If an error occurs in the sequence of VISA VIs that execute in the program, the Simple Error Handler will pop up a dialogue box informing the user of the error and the text message associated with the VISA Error Code.

VISA Attributes

The basic operations that are associated with message and register-based resources in VISA have now been introduced. These operations allow register access, and message-

based communication. In addition to the basic communication operations, VISA resources have a variety of attributes whose values can be read or set in a program.

In a LabVIEW program, these attributes are handled programmatically in the same way that the properties of front panel controls and indicators are handled. Attribute nodes are used to read or set the values of VISA attributes. A VISA attribute node is shown in the figure below.



VISA Attribute Node

Get and/or set the indicated attributes. This node is growable and evaluation starts from the top proceeding down the node until the final attribute or until an error occurs.

The VISA attribute node contains a single attribute terminal when it is placed initially on the block diagram. However, it can be resized to contain as many different terminals as necessary. The initial terminal on the VISA attribute node is a read terminal. This means that the value of the attribute selected in that terminal will be read. This is indicated by the small arrow pointing to the right at the right edge of the terminal. Terminals can individually be changed from a read terminal to a write terminal by popping-up on the attribute you wish to change.

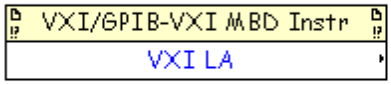
Note: *Popping up on the terminals sometimes will not give you the option of changing the terminal to a read or to a write. This is due to the fact that some attributes are read only. Their value can not be set.*

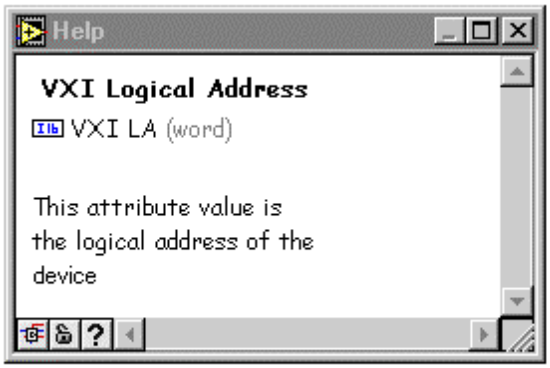
To select the attribute in each terminal of the attribute node, pop up on the attribute node terminal and choose "Select Item." This will provide a list of all the possible attributes that can be set in the program. This will produce a long list of attributes. The number of different attributes that are shown under the Select Item choice of the VISA Attribute Node can be limited. To do this, the VISA Class of the attribute node can be changed.

To change the VISA class pop up on the VISA attribute node and go to the VISA Class selection. Several different classes can be selected under this option besides the default INSTR class which encompasses all the VISA attributes. These classes simply limit the attributes displayed to those related to that class instead of all the VISA attributes. Once a session is connected to the Session input terminal of the attribute, the VISA Class will be set to the class associated with that session.

Initially, the VISA attributes will be somewhat unfamiliar and their exact nature may not be clear from the name alone. The VISA online reference contains more detailed

descriptions of the various attributes and their values. The LabVIEW online reference also contains information on the attributes. Brief descriptions of individual attributes are also available in the simple help window. To get a brief description of a specific attribute, select the attribute in one of the terminals of an attribute node and then open the simple help window. This is shown for the VXI LA attribute below.





Note that the help window shows the specific variable type of the attribute and gives a brief description of what the attribute does. In cases where it is not clear what variable type to use for reading or writing an attribute, remember popping up on an attribute node and selecting Create Constant, Create Control, or Create Indicator will automatically select the appropriate variable type.

There are two basic types of VISA attributes - Global Attributes and Local Attributes. Global attributes are specific to a resource while Local Attributes are specific to a session. For example, the VXI LA attribute that is shown in the figure above is an example of a global attribute. It applies to all of the sessions that are open to that resource. A local attribute is an attribute that could be different for individual sessions to a specific resource. An example of a local attribute is the time out value. Some of the common attributes for each resource type are shown in the list below.

Serial

Serial Baud Rate – The baud rate for the serial port.

Serial Data Bits – The number data bits used for serial transmissions.

Serial Parity – The parity used for serial transmissions.

Serial Stop Bits – The number of stop bits used for serial transmissions.

GPIB

Gpib Readdressing – Specifies if the device should be readressed before every write operation.

Gpib Unaddressing – Specifies if the device should be unaddressed after read and write operations.

VXI

Mainframe Logical Address - The lowest logical address of a device in the same chassis with the resource.

Manufacturer Identification - The manufacturer ID number from the device's configuration registers.

Model Code - The model code of the device from the device's configuration registers.

Slot - The slot in the chassis that the resource resides in.

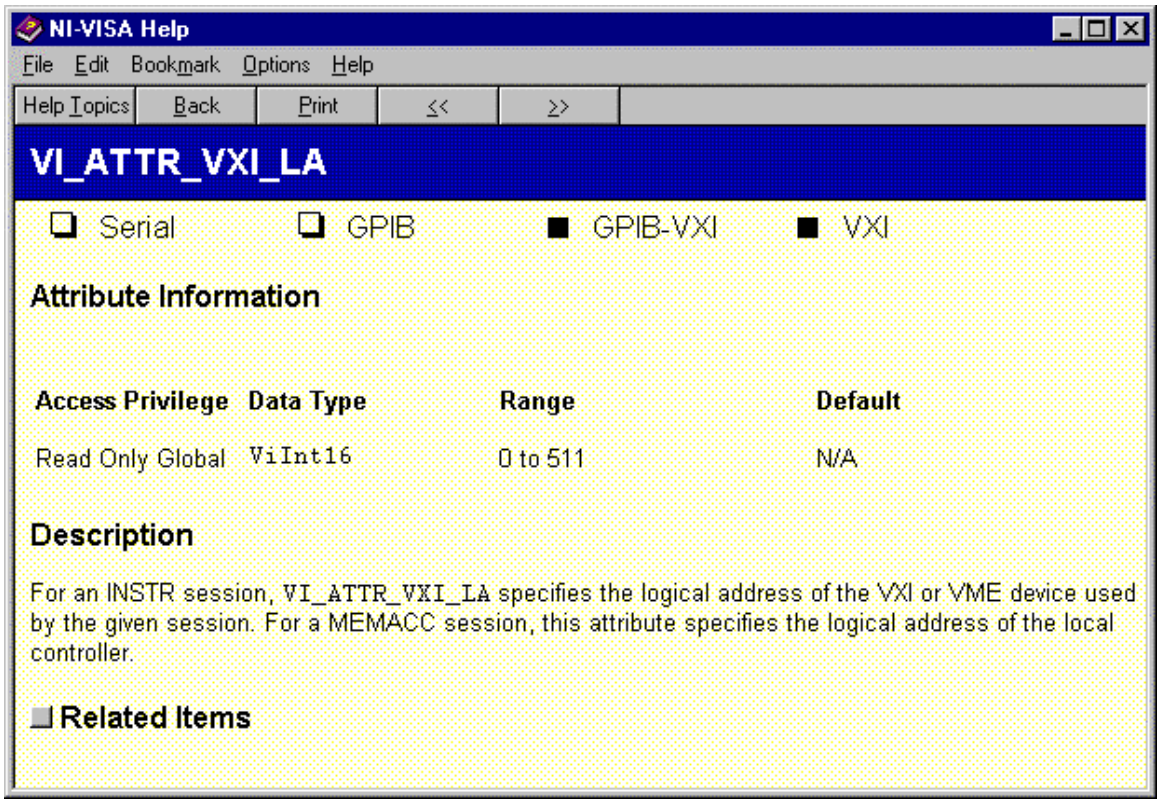
VXI Logical Address - The logical address of the device.

VXI Memory Address Space - The VXI address space used by the resource.

VXI Memory Address Base - The base address of the memory region used by the resource.

VXI Memory Address Size - The size of memory region used by the resource.

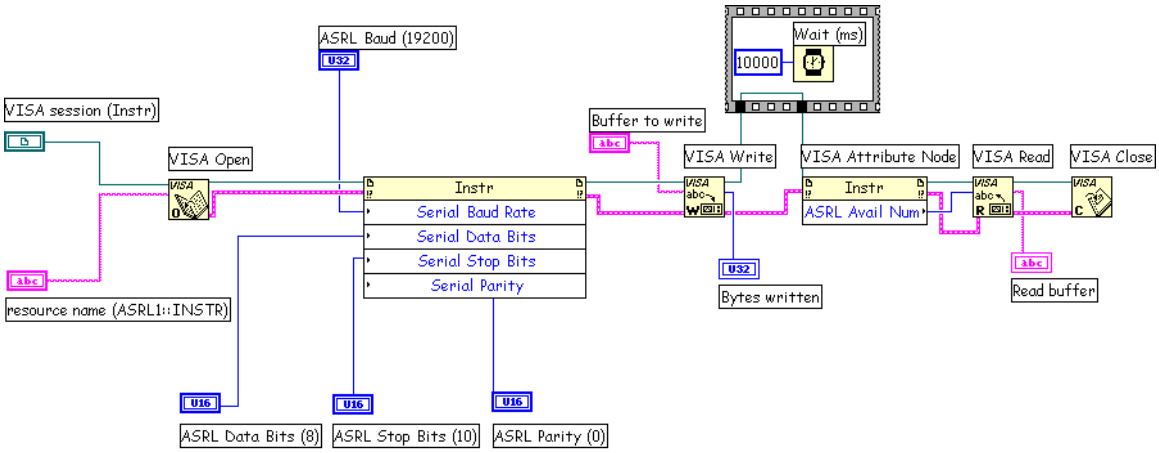
There are many other attributes besides those given in this list. There are also attributes that are not specific to a certain interface type. The timeout attribute which is the timeout used in message-based I/O operations is a good example of such an attribute. The most complete source for information about attributes is the VISA online reference (found under the VXI PNP program group). This information is found under the Attributes topic in the online reference. The entry for the VXI Logical Address attribute in the VISA Online Reference is shown in the figure below.



The VISA online help shows which type of interfaces the attribute applies to, if the attribute is local or global, its data type, and what the valid range of values are for the attribute. It also shows related items and gives a detailed description of the attribute. However, like the VISAIC utility, the VISA online help is not installed by the default LabVIEW installation. To obtain it, VISA must be reinstalled.

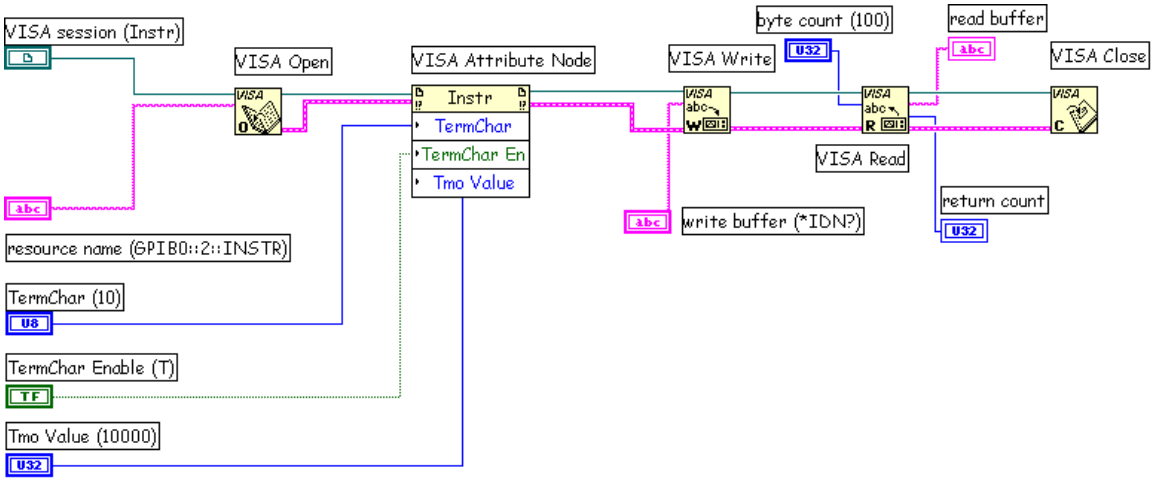
VISA Attribute Examples

This section contains three simple examples of using attributes in VISA programs. The first, shown in the figure below, writes a string to a serial instrument and reads the response.



The VI opens a session to the serial port COM 1 and initializes it for 19200 baud, 8 data bits, no parity, and 1 stop bit. A string is then written out the port. After writing the string and waiting for 10 seconds, the number of bytes that have been returned by the device are obtained using another VISA attribute. These bytes are then read from the port. Notice that in this example the value used to set the number of stop bits to one is actually the value 10.

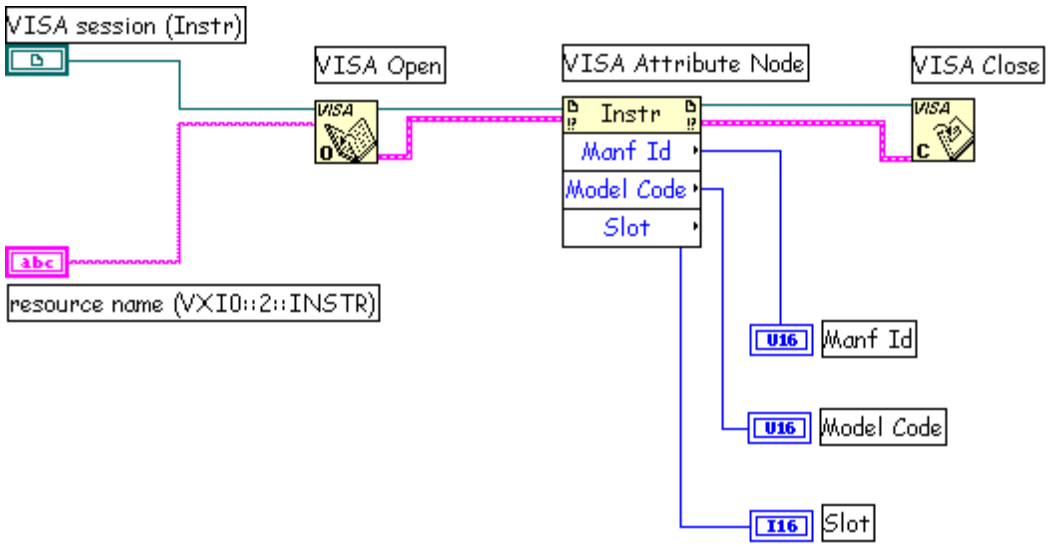
The next example shows how to use attributes to set a termination character for VISA read operations. Some message-based devices send a special character termination character when they have no more data to send.



This VI opens a session to a GPIB instrument at primary address two. The VI sets the termination character to a linefeed (decimal value 10) and then enables the use of a termination character with another attribute. The VI also sets the timeout attribute to 10000 milliseconds (10 seconds). It then writes the string *IDN? to the instrument and tries to read back a 100 character response. The read will terminate when the termination

character is received. If the termination character is not received, it will terminate after 100 bytes are received or timeout after 10 seconds.

The final example shows how to read some of the common attributes associated with a VXI instrument.



This VI opens a session to a VXI instrument at logical address 2 and reads the Manufacturer Id, Model Code, and Slot for the VXI module.

Debugging A VISA Problem

This section contains information on debugging problems with VISA programs. Because of VISA's nature, there are more possibilities to consider when debugging VISA problems than when working with standalone drivers. VISA makes calls into NI-VXI, NI-488, or Operating System serial API's. Therefore, problems that appear in VISA could be related to the driver VISA is calling, and not VISA itself.

If no VISA VIs appear to be working in LabVIEW (including instrument drivers), one of the first things debugging steps is to try is the basic VISA Find Resources VI. This VI will run without any other VISA VI's in the block diagram. If this VI produces strange errors such as error codes that are not standard VISA errors, the problem is most likely that the wrong version of VISA is installed or that VISA is not installed correctly. If VISA Find Resources runs correctly, there is not a problem with LabVIEW interfacing with the VISA driver. The next step is to narrow down what sequence of VI's are producing the error in the LabVIEW program.

If it is a simple sequence of events that is producing the error, a good next step in debugging is to try the same sequence interactively with the VISAIC utility. It is

generally a good idea to do initial program development interactively. If the interactive utility works successfully but the same sequence in LabVIEW does not, it is an indication that LabVIEW's implementation of VISA may have a problem. If the same sequence exhibits the same problem interactively in VISAIC it is possible that a problem exists with one of the drivers VISA is calling. The interactive utilities for these drivers (VIC for NI-VXI and IBIC for NI-488.2) can be used to try to perform the equivalent operations. If the problems persist on this level it is an indication that there may be a problem with the lower level driver or its installation. This is only a very rough outline of debugging a VISA problem. The main point to remember when debugging is that VISA programs involve several layers.