

### Modbus LabVIEW Library Help

February 2009, 372767A-01

This help file describes the Modbus LabVIEW VI library.

For more information about this help file, refer to the following topics:

Using Help

**Important Information** 

Technical Support and Professional Services

To comment on National Instruments documentation, refer to the <u>National</u> <u>Instruments Web site</u>.

© 2009 National Instruments Corporation. All rights reserved.

### **Using Help**

<u>Conventions</u> <u>Navigating Help</u> <u>Searching Help</u> <u>Printing Help File Topics</u>

### Conventions

This help file uses the following formatting and typographical conventions:

$\mathbf{N}$	This icon denotes a note, which alerts you to important information.
P	This icon denotes a tip, which alerts you to advisory information.
bold	Bold text denotes items that you must select or click in the software, such as menu items and dialog box options. Bold text also denotes parameter names.
<u>green</u>	Underlined text in this color denotes a link to a help topic, help file, or Web address.
<u>purple</u>	Underlined text in this color denotes a visited link to a help topic, help file, or Web address.
italic	Italic text denotes variables, emphasis, cross-references, or an introduction to a key concept. Italic text also denotes text that is a placeholder for a word or value that you must supply.
monospace	Text in this font denotes text or characters that you should enter from the keyboard, sections of code, programming examples, and syntax examples. This font is also used for the proper names of disk drives, paths, directories,

programs, subprograms, subroutines, device names, functions, operations, variables, filenames, and extensions.

## **Navigating Help (Windows Only)**

To navigate this help file, use the **Contents**, **Index**, and **Search** tabs to the left of this window or use the following toolbar buttons located above the tabs:



**Note** The information in this topic only applies to the help file installed with the Windows version of the software.

- Hide—Hides the navigation pane from view.
- Locate—Locates the currently displayed topic in the Contents tab, allowing you to view related topics.
- **Back**—Displays the previously viewed topic.
- Forward—Displays the topic you viewed before clicking the **Back** button.
- **Options**—Displays a list of commands and viewing options for the help file.

## **Searching Help (Windows Only)**

Use the **Search** tab to the left of this window to locate content in this help file.



**Note** The information in this topic only applies to the help file installed with the Windows version of the software.

If you want to search for words in a certain order, such as "related documentation," add quotation marks around the search words as shown in the example. Searching for terms on the **Search** tab allows you to quickly locate specific information and information in topics that are not included on the **Contents** tab.



**Tip** You can temporarily disable the search highlighting feature by selecting **Options**»**Search Highlight Off** from the toolbar.

#### Wildcards

You also can search using asterisk (\*) or question mark (?) wildcards. Use the asterisk wildcard to return topics that contain a certain string. For example, a search for "prog\*" lists topics that contain the words "program," "programmatically," "progress," and so on.

Use the question mark wildcard as a substitute for a single character in a search term. For example, "?ext" lists topics that contain the words "next," "text," and so on.



**Note** Wildcard searching will not work on Simplified Chinese, Traditional Chinese, Japanese, and Korean systems.

#### **Nested Expressions**

Use nested expressions to combine searches to further refine a search. You can use Boolean expressions and wildcards in a nested expression. For example, "example AND (program OR VI)" lists topics that contain "example program" or "example VI." You cannot nest expressions more than five levels.

#### **Boolean Expressions**

Click the **•** button to add Boolean expressions to a search. The following Boolean operators are available:

- **AND** (default)—Returns topics that contain both search terms. You do not need to specify this operator unless you are using nested expressions.
- **OR**—Returns topics that contain either the first or second term.
- **NOT**—Returns topics that contain the first term without the second term.
- **NEAR**—Returns topics that contain both terms within eight words of each other.

#### **Search Options**

Use the following checkboxes on the **Search** tab to customize a search:

- **Search previous results**—Narrows the results from a search that returned too many topics. You must remove the checkmark from this checkbox to search all topics.
- Match similar words—Broadens a search to return topics that contain words similar to the search terms. For example, a search for "program" lists topics that include the words "programs," "programming," and so on.
- Search titles only—Searches only in the titles of topics.

### **Printing Help File Topics (Windows Only)**

Complete the following steps to print an entire book from the **Contents** tab:



**Note** The information in this topic only applies to the help file installed with the Windows version of the software.

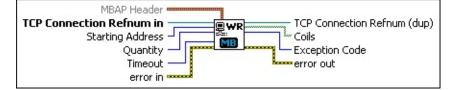
- 1. Right-click the book.
- 2. Select **Print** from the shortcut menu to display the **Print Topics** dialog box.
- 3. Select the **Print the selected heading and all subtopics** option.
  - Note Select Print the selected topic if you want to print the single topic you have selected in the **Contents** tab.
- 4. Click the **OK** button.

#### **Printing PDF Documents**

This help file may contain links to PDF documents. To print PDF documents, click the print button located on the Adobe Acrobat Viewer toolbar.

### MB Ethernet Master Query Read Coils (poly).vi

The master uses this VI read the slaves' coils. This corresponds to a public function code of 1 in the MODBUS protocol.



**MBAP Header** is a cluster containing the transaction ID and Unit ID. This is additional information not usually required for a MODBUS transaction.

**Transaction Identifier**—This identifier is for transaction pairing; the MODBUS server copies the request transaction identifier in the response.

**Unit Identifier**—This field is for intra-system routing purposes. It typically communicates to a MODBUS serial line slave through a gateway between an Ethernet TCP-IP network and a MODBUS serial line. The MODBUS client sets this field in the request, and the server response must return this field with the same value.

- **TCP Connection Refnum in** is a network connection refnum that uniquely identifies the TCP connection. Use TCP Open Connection.vi to open a TCP connection.
- **Starting Address** is the first address location of the coil to read. This address is sometimes referred to as *offset*. Refer to your device's documentation to find the coil address mapping.

Be sure to specify the correct address in the MODBUS device configuration software for the register you want to use in LabVIEW. In most MODBUS device configuration software, you must enter a name for the register you want to use. Per MODBUS convention, the register address of the slave device is calculated by subtracting 1 from the register name that you specify in the master device configuration software. The MODBUS LabVIEW library expects register addresses, not register names, so you may need to subtract 1 from the address you defined in the MODBUS device configuration software. For example, a register name defined as 2 in a MODBUS configuration device translates to register address 1 in the Holding Registers table of the LabVIEW MODBUS library, as shown below.

MODBUS Holding Register Name = 2 Device

LabVIEW Holding Register Address = 1

- Quantity represents how many coils to read from the slave. The VI returns the coil at Starting Address and each following coil up to Quantity. For example, if Starting Address is 0 and Quantity is 4, the VI reads coils from the slave at address 0, 1, 2, and 3.
- **Timeout** specifies the maximum time period, in milliseconds, that the VI waits for the slave's response before throwing an error.
- error in describes error conditions that occur before this VI or function runs. The default is no error. If an error occurred before this VI or function runs, the VI or function passes the error in value to error out. If an error occurs while this VI or function runs, the VI or function runs normally and sets its own error status in error out. Use the Simple Error Handler or General Error Handler VIs to display the description of the error code. Use error in and error out to check errors and to specify execution order by wiring error out from one node to error in of the next node.
  - **status** is TRUE (X) if an error occurred before this VI or function ran or FALSE (checkmark) to indicate a warning or that no error occurred before this VI or function ran. The default is FALSE.
  - **code** is the error or warning code. The default is 0. If **status** is TRUE, **code** is a negative error code. If **status** is FALSE, **code** is 0 or a warning code.
  - **source** identifies where an error occurred. The source string includes the name of the VI that produced the error, what inputs are in error, and how to eliminate the error.
- TCP Connection Refnum (dup) can be an input to another MODBUS VI, or you can close it using TCP Close Connection.vi.
- **[TF]** Coils represents the data read from the slave.

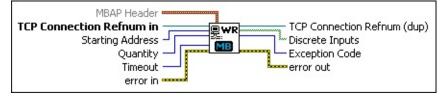
**Exception Code** is nonzero if a MODBUS error has occurred. The exception code mapping is as follows:

- 0 No error
- 1 Illegal function
- 2 Illegal data address
- 3 Illegal data value
- 4 Failure in associated device
- 5 Acknowledge
- 6 Busy, rejected message
- 7 NAK—Negative acknowledge

- **error out** contains error information. If **error in** indicates that an error occurred before this VI or function ran, **error out** contains the same error information. Otherwise, it describes the error status that this VI or function produces. Right-click the **error out** front panel indicator and select **Explain Error** from the shortcut menu for more information about the error.
  - **status** is TRUE (X) if an error occurred or FALSE (checkmark) to indicate a warning or that no error occurred.
  - **code** is the error or warning code. If **status** is TRUE, **code** is a nonzero error code. If **status** is FALSE, **code** is 0 or a warning code.
  - **source** describes the origin of the error or warning and is, in most cases, the name of the VI or function that produced the error or warning.

# MB Ethernet Master Query Read Discrete Inputs (poly).vi

The master uses this VI to read the slaves' discrete inputs. This corresponds to a public function code of 2 in the MODBUS protocol.



**MBAP Header** is a cluster containing the transaction ID and Unit ID. This is additional information not usually required for a MODBUS transaction.

**Transaction Identifier**—This identifier is for transaction pairing; the MODBUS server copies the request transaction identifier in the response.

**Unit Identifier**—This field is for intra-system routing purposes. It typically communicates to a MODBUS serial line slave through a gateway between an Ethernet TCP-IP network and a MODBUS serial line. The MODBUS client sets this field in the request, and the server response must return this field with the same value.

- **TCP Connection Refnum in** is a network connection refnum that uniquely identifies the TCP connection. Use TCP Open Connection.vi to open a TCP connection.
- **Starting Address** is the first address location of the discrete inputs to read. This address is sometimes referred to as *offset*. Refer to your device's documentation to find the discrete input address mapping.

Be sure to specify the correct address in the MODBUS device configuration software for the register you want to use in LabVIEW. In most MODBUS device configuration software, you must enter a name for the register you want to use. Per MODBUS convention, the register address of the slave device is calculated by subtracting 1 from the register name that you specify in the master device configuration software. The MODBUS LabVIEW library expects register addresses, not register names, so you may need to subtract 1 from the address you defined in the MODBUS device configuration software. For example, a register name defined as 2 in a MODBUS configuration device translates to register address 1 in the Holding Registers table of the LabVIEW MODBUS library, as shown below.

MODBUS Holding Register Name = 2 Device

LabVIEW Holding Register Address = 1

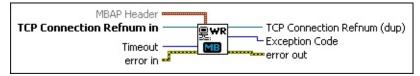
- Quantity represents how many discrete inputs to read from the slave. The VI returns the discrete input at **Starting Address** and each following discrete input up to **Quantity**. For example, if **Starting Address** is 0 and **Quantity** is 4, the VI reads discrete inputs from the slave at address 0, 1, 2, and 3.
- **Timeout** specifies the maximum time period, in milliseconds, that the VI waits for the slave's response before throwing an error.
- error in describes error conditions that occur before this VI or function runs. The default is no error. If an error occurred before this VI or function runs, the VI or function passes the error in value to error out. If an error occurs while this VI or function runs, the VI or function runs normally and sets its own error status in error out. Use the Simple Error Handler or General Error Handler VIs to display the description of the error code. Use error in and error out to check errors and to specify execution order by wiring error out from one node to error in of the next node.
  - **status** is TRUE (X) if an error occurred before this VI or function ran or FALSE (checkmark) to indicate a warning or that no error occurred before this VI or function ran. The default is FALSE.
  - **code** is the error or warning code. The default is 0. If **status** is TRUE, **code** is a negative error code. If **status** is FALSE, **code** is 0 or a warning code.
  - **source** identifies where an error occurred. The source string includes the name of the VI that produced the error, what inputs are in error, and how to eliminate the error.

- TCP Connection Refnum (dup) can be an input to another MODBUS VI, or you can close it using TCP Close Connection.vi.
- **[TF] Discrete Inputs** represents the data read from the slave.
- **Exception Code** is nonzero if a MODBUS error has occurred. The exception code mapping is as follows:
  - 0 No error
  - 1 Illegal function
  - 2 Illegal data address
  - 3 Illegal data value
  - 4 Failure in associated device
  - 5 Acknowledge
  - 6 Busy, rejected message
  - 7 NAK—Negative acknowledge

- **error out** contains error information. If **error in** indicates that an error occurred before this VI or function ran, **error out** contains the same error information. Otherwise, it describes the error status that this VI or function produces. Right-click the **error out** front panel indicator and select **Explain Error** from the shortcut menu for more information about the error.
  - **status** is TRUE (X) if an error occurred or FALSE (checkmark) to indicate a warning or that no error occurred.
  - **code** is the error or warning code. If **status** is TRUE, **code** is a nonzero error code. If **status** is FALSE, **code** is 0 or a warning code.
  - **source** describes the origin of the error or warning and is, in most cases, the name of the VI or function that produced the error or warning.

## MB Ethernet Master Query Read Exception Status (poly).vi

The master uses this VI to read the exception status from the slave. This corresponds to a public function code of 7 in the MODBUS protocol.



**MBAP Header** is a cluster containing the transaction ID and Unit ID. This is additional information not usually required for a MODBUS transaction.

**Transaction Identifier**—This identifier is for transaction pairing; the MODBUS server copies the request transaction identifier in the response.

**Unit Identifier**—This field is for intra-system routing purposes. It typically communicates to a MODBUS serial line slave through a gateway between an Ethernet TCP-IP network and a MODBUS serial line. The MODBUS client sets this field in the request, and the server response must return this field with the same value.

- **TCP Connection Refnum in** is a network connection refnum that uniquely identifies the TCP connection. Use TCP Open Connection.vi to open a TCP connection.
- **Timeout** specifies the maximum time period, in milliseconds, that the VI waits for the slave's response before throwing an error.
- error in describes error conditions that occur before this VI or function runs. The default is no error. If an error occurred before this VI or function runs, the VI or function passes the error in value to error out. If an error occurs while this VI or function runs, the VI or function runs normally and sets its own error status in error out. Use the Simple Error Handler or General Error Handler VIs to display the description of the error code. Use error in and error out to check errors and to specify execution order by wiring error out from one node to error in of the next node.

**status** is TRUE (X) if an error occurred before this VI or

function ran or FALSE (checkmark) to indicate a warning or that no error occurred before this VI or function ran. The default is FALSE.

- **code** is the error or warning code. The default is 0. If **status** is TRUE, **code** is a negative error code. If **status** is FALSE, **code** is 0 or a warning code.
- **source** identifies where an error occurred. The source string includes the name of the VI that produced the error, what inputs are in error, and how to eliminate the error.
- TCP Connection Refnum (dup) can be an input to another MODBUS VI, or you can close it using TCP Close Connection.vi.
- **Exception Code** is nonzero if a MODBUS error has occurred. The exception code mapping is as follows:
  - 0 No error
  - 1 Illegal function
  - 2 Illegal data address
  - 3 Illegal data value
  - 4 Failure in associated device
  - 5 Acknowledge
  - 6 Busy, rejected message
  - 7 NAK—Negative acknowledge

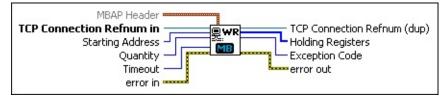
- **error out** contains error information. If **error in** indicates that an error occurred before this VI or function ran, **error out** contains the same error information. Otherwise, it describes the error status that this VI or function produces. Right-click the **error out** front panel indicator and select **Explain Error** from the shortcut menu for more information about the error.
  - **status** is TRUE (X) if an error occurred or FALSE (checkmark) to indicate a warning or that no error occurred.
  - **code** is the error or warning code. If **status** is TRUE, **code** is a nonzero error code. If **status** is FALSE, **code** is 0 or a

warning code.

**source** describes the origin of the error or warning and is, in most cases, the name of the VI or function that produced the error or warning.

# MB Ethernet Master Query Read Holding Registers (poly).vi

The master uses this VI to read the slaves' holding registers. This corresponds to a public function code of 3 in the MODBUS protocol.



**MBAP Header** is a cluster containing the transaction ID and Unit ID. This is additional information not usually required for a MODBUS transaction.

**Transaction Identifier**—This identifier is for transaction pairing; the MODBUS server copies the request transaction identifier in the response.

**Unit Identifier**—This field is for intra-system routing purposes. It typically communicates to a MODBUS serial line slave through a gateway between an Ethernet TCP-IP network and a MODBUS serial line. The MODBUS client sets this field in the request, and the server response must return this field with the same value.

- **TCP Connection Refnum in** is a network connection refnum that uniquely identifies the TCP connection. Use TCP Open Connection.vi to open a TCP connection.
- **Starting Address** is the first address location of the holding register to read. This address is sometimes referred to as *offset*. Refer to your device's documentation to find the holding register address mapping.

Be sure to specify the correct address in the MODBUS device configuration software for the register you want to use in LabVIEW. In most MODBUS device configuration software, you must enter a name for the register you want to use. Per MODBUS convention, the register address of the slave device is calculated by subtracting 1 from the register name that you specify in the master device configuration software. The MODBUS LabVIEW library expects register addresses, not register names, so you may need to subtract 1 from the address you defined in the MODBUS device configuration software. For example, a register name defined as 2 in a MODBUS configuration device translates to register address 1 in the Holding Registers table of the LabVIEW MODBUS library, as shown below.

MODBUS Holding Register Name = 2 Device

LabVIEW Holding Register Address = 1

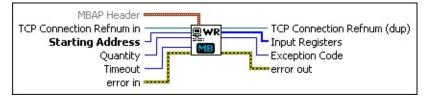
- Quantity represents how many holding registers to read from the slave. The VI returns the holding register at **Starting Address** and each following holding register up to **Quantity**. For example, if **Starting Address** is 0 and **Quantity** is 4, the VI reads holding registers from the slave at address 0, 1, 2, and 3.
- **Timeout** specifies the maximum time period, in milliseconds, that the VI waits for the slave's response before throwing an error.
- error in describes error conditions that occur before this VI or function runs. The default is no error. If an error occurred before this VI or function runs, the VI or function passes the error in value to error out. If an error occurs while this VI or function runs, the VI or function runs normally and sets its own error status in error out. Use the Simple Error Handler or General Error Handler VIs to display the description of the error code. Use error in and error out to check errors and to specify execution order by wiring error out from one node to error in of the next node.
  - **status** is TRUE (X) if an error occurred before this VI or function ran or FALSE (checkmark) to indicate a warning or that no error occurred before this VI or function ran. The default is FALSE.
  - **code** is the error or warning code. The default is 0. If **status** is TRUE, **code** is a negative error code. If **status** is FALSE, **code** is 0 or a warning code.
  - **source** identifies where an error occurred. The source string includes the name of the VI that produced the error, what inputs are in error, and how to eliminate the error.

- TCP Connection Refnum (dup) can be an input to another MODBUS VI, or you can close it using TCP Close Connection.vi.
- **Holding Registers** represents the data read from the slave.
- **Exception Code** is nonzero if a MODBUS error has occurred. The exception code mapping is as follows:
  - 0 No error
  - 1 Illegal function
  - 2 Illegal data address
  - 3 Illegal data value
  - 4 Failure in associated device
  - 5 Acknowledge
  - 6 Busy, rejected message
  - 7 NAK—Negative acknowledge

- **error out** contains error information. If **error in** indicates that an error occurred before this VI or function ran, **error out** contains the same error information. Otherwise, it describes the error status that this VI or function produces. Right-click the **error out** front panel indicator and select **Explain Error** from the shortcut menu for more information about the error.
  - **status** is TRUE (X) if an error occurred or FALSE (checkmark) to indicate a warning or that no error occurred.
  - **code** is the error or warning code. If **status** is TRUE, **code** is a nonzero error code. If **status** is FALSE, **code** is 0 or a warning code.
  - **source** describes the origin of the error or warning and is, in most cases, the name of the VI or function that produced the error or warning.

# MB Ethernet Master Query Read Input Registers (poly).vi

The master uses this VI to read the slaves' input registers. This corresponds to a public function code of 4 in the MODBUS protocol.



**MBAP Header** is a cluster containing the transaction ID and Unit ID. This is additional information not usually required for a MODBUS transaction.

**Transaction Identifier**—This identifier is for transaction pairing; the MODBUS server copies the request transaction identifier in the response.

**Unit Identifier**—This field is for intra-system routing purposes. It typically communicates to a MODBUS serial line slave through a gateway between an Ethernet TCP-IP network and a MODBUS serial line. The MODBUS client sets this field in the request, and the server response must return this field with the same value.

- **TCP Connection Refnum in** is a network connection refnum that uniquely identifies the TCP connection. Use TCP Open Connection.vi to open a TCP connection.
- **Starting Address** is the first address location of the input register to read. This address is sometimes referred to as *offset*. Refer to your device's documentation to find the input register address mapping.

Be sure to specify the correct address in the MODBUS device configuration software for the register you want to use in LabVIEW. In most MODBUS device configuration software, you must enter a name for the register you want to use. Per MODBUS convention, the register address of the slave device is calculated by subtracting 1 from the register name that you specify in the master device configuration software. The MODBUS LabVIEW library expects register addresses, not register names, so you may need to subtract 1 from the address you defined in the MODBUS device configuration software. For example, a register name defined as 2 in a MODBUS configuration device translates to register address 1 in the Holding Registers table of the LabVIEW MODBUS library, as shown below.

MODBUS Holding Register Name = 2 Device

LabVIEW Holding Register Address = 1

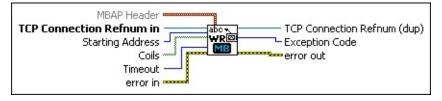
- Quantity represents how many input registers to read from the slave. The VI returns the input register at **Starting Address** and each following input register up to **Quantity**. For example, if **Starting Address** is 0 and **Quantity** is 4, the VI reads input registers from the slave at address 0, 1, 2, and 3.
- **Timeout** specifies the maximum time period, in milliseconds, that the VI waits for the slave's response before throwing an error.
- error in describes error conditions that occur before this VI or function runs. The default is no error. If an error occurred before this VI or function runs, the VI or function passes the error in value to error out. If an error occurs while this VI or function runs, the VI or function runs normally and sets its own error status in error out. Use the Simple Error Handler or General Error Handler VIs to display the description of the error code. Use error in and error out to check errors and to specify execution order by wiring error out from one node to error in of the next node.
  - **status** is TRUE (X) if an error occurred before this VI or function ran or FALSE (checkmark) to indicate a warning or that no error occurred before this VI or function ran. The default is FALSE.
  - **code** is the error or warning code. The default is 0. If **status** is TRUE, **code** is a negative error code. If **status** is FALSE, **code** is 0 or a warning code.
  - **source** identifies where an error occurred. The source string includes the name of the VI that produced the error, what inputs are in error, and how to eliminate the error.

- TCP Connection Refnum (dup) can be an input to another MODBUS VI, or you can close it using TCP Close Connection.vi.
- **Input Registers** represents the data read from the slave.
- **Exception Code** is nonzero if a MODBUS error has occurred. The exception code mapping is as follows:
  - 0 No error
  - 1 Illegal function
  - 2 Illegal data address
  - 3 Illegal data value
  - 4 Failure in associated device
  - 5 Acknowledge
  - 6 Busy, rejected message
  - 7 NAK—Negative acknowledge

- **error out** contains error information. If **error in** indicates that an error occurred before this VI or function ran, **error out** contains the same error information. Otherwise, it describes the error status that this VI or function produces. Right-click the **error out** front panel indicator and select **Explain Error** from the shortcut menu for more information about the error.
  - **status** is TRUE (X) if an error occurred or FALSE (checkmark) to indicate a warning or that no error occurred.
  - **code** is the error or warning code. If **status** is TRUE, **code** is a nonzero error code. If **status** is FALSE, **code** is 0 or a warning code.
  - **source** describes the origin of the error or warning and is, in most cases, the name of the VI or function that produced the error or warning.

# MB Ethernet Master Query Write Multiple Coils (poly).vi

The master uses this VI to write multiple coils to the slave. This corresponds to a public function code of 15 in the MODBUS protocol.



**MBAP Header** is a cluster containing the transaction ID and Unit ID. This is additional information not usually required for a MODBUS transaction.

**Transaction Identifier**—This identifier is for transaction pairing; the MODBUS server copies the request transaction identifier in the response.

**Unit Identifier**—This field is for intra-system routing purposes. It typically communicates to a MODBUS serial line slave through a gateway between an Ethernet TCP-IP network and a MODBUS serial line. The MODBUS client sets this field in the request, and the server response must return this field with the same value.

- **TCP Connection Refnum in** is a network connection refnum that uniquely identifies the TCP connection. Use TCP Open Connection.vi to open a TCP connection.
- **Starting Address** is the address location of the first coil to write. This address is sometimes referred to as *offset*. Refer to your device's documentation to find the coil address mapping.

Be sure to specify the correct address in the MODBUS device configuration software for the register you want to use in LabVIEW. In most MODBUS device configuration software, you must enter a name for the register you want to use. Per MODBUS convention, the register address of the slave device is calculated by subtracting 1 from the register name that you specify in the master device configuration software. The MODBUS LabVIEW library expects register addresses, not register names, so you may need to subtract 1 from the address you defined in the MODBUS device configuration software. For example, a register name defined as 2 in a MODBUS configuration device translates to register address 1 in the Holding Registers table of the LabVIEW MODBUS library, as shown below.

```
MODBUS Holding Register Name = 2
Device
```

```
LabVIEW Holding Register Address = 1
```

- **Coils** is the coils to write. The array length represents the number of coils that will be written.
- **Timeout** specifies the maximum time period, in milliseconds, that the VI waits for the slave's response before throwing an error.
- error in describes error conditions that occur before this VI or function runs. The default is no error. If an error occurred before this VI or function runs, the VI or function passes the error in value to error out. If an error occurs while this VI or function runs, the VI or function runs normally and sets its own error status in error out. Use the Simple Error Handler or General Error Handler VIs to display the description of the error code. Use error in and error out to check errors and to specify execution order by wiring error out from one node to error in of the next node.
  - **status** is TRUE (X) if an error occurred before this VI or function ran or FALSE (checkmark) to indicate a warning or that no error occurred before this VI or function ran. The default is FALSE.
  - **code** is the error or warning code. The default is 0. If **status** is TRUE, **code** is a negative error code. If **status** is FALSE, **code** is 0 or a warning code.
  - **source** identifies where an error occurred. The source string includes the name of the VI that produced the error, what inputs are in error, and how to eliminate the error.
- TCP Connection Refnum (dup) can be an input to another MODBUS VI, or you can close it using TCP Close Connection.vi.

**Exception Code** is nonzero if a MODBUS error has occurred. The

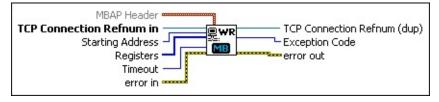
exception code mapping is as follows:

- 0 No error
- 1 Illegal function
- 2 Illegal data address
- 3 Illegal data value
- 4 Failure in associated device
- 5 Acknowledge
- 6 Busy, rejected message
- 7 NAK—Negative acknowledge

- error out contains error information. If error in indicates that an error occurred before this VI or function ran, error out contains the same error information. Otherwise, it describes the error status that this VI or function produces. Right-click the error out front panel indicator and select Explain Error from the shortcut menu for more information about the error.
  - **status** is TRUE (X) if an error occurred or FALSE (checkmark) to indicate a warning or that no error occurred.
  - **code** is the error or warning code. If **status** is TRUE, **code** is a nonzero error code. If **status** is FALSE, **code** is 0 or a warning code.
  - **source** describes the origin of the error or warning and is, in most cases, the name of the VI or function that produced the error or warning.

### MB Ethernet Master Query Write Multiple Registers (poly).vi

The master uses this VI to write multiple registers to the slave. This corresponds to a public function code of 16 in the MODBUS protocol.



**MBAP Header** is a cluster containing the transaction ID and Unit ID. This is additional information not usually required for a MODBUS transaction.

**Transaction Identifier**—This identifier is for transaction pairing; the MODBUS server copies the request transaction identifier in the response.

**Unit Identifier**—This field is for intra-system routing purposes. It typically communicates to a MODBUS serial line slave through a gateway between an Ethernet TCP-IP network and a MODBUS serial line. The MODBUS client sets this field in the request, and the server response must return this field with the same value.

- **TCP Connection Refnum in** is a network connection refnum that uniquely identifies the TCP connection. Use TCP Open Connection.vi to open a TCP connection.
- **Starting Address** is the address location of the first register to write. This address is sometimes referred to as *offset*. Refer to your device's documentation to find the register address mapping.

Be sure to specify the correct address in the MODBUS device configuration software for the register you want to use in LabVIEW. In most MODBUS device configuration software, you must enter a name for the register you want to use. Per MODBUS convention, the register address of the slave device is calculated by subtracting 1 from the register name that you specify in the master device configuration software. The MODBUS LabVIEW library expects register addresses, not register names, so you may need to subtract 1 from the address you defined in the MODBUS device configuration software. For example, a register name defined as 2 in a MODBUS configuration device translates to register address 1 in the Holding Registers table of the LabVIEW MODBUS library, as shown below.

```
MODBUS Holding Register Name = 2
Device
```

```
LabVIEW Holding Register Address = 1
```

- **Registers** are the registers to write. The array length represents how many registers will be written.
- **Timeout** specifies the maximum time period, in milliseconds, that the VI waits for the slave's response before throwing an error.
- error in describes error conditions that occur before this VI or function runs. The default is no error. If an error occurred before this VI or function runs, the VI or function passes the error in value to error out. If an error occurs while this VI or function runs, the VI or function runs normally and sets its own error status in error out. Use the Simple Error Handler or General Error Handler VIs to display the description of the error code. Use error in and error out to check errors and to specify execution order by wiring error out from one node to error in of the next node.
  - **status** is TRUE (X) if an error occurred before this VI or function ran or FALSE (checkmark) to indicate a warning or that no error occurred before this VI or function ran. The default is FALSE.
  - **code** is the error or warning code. The default is 0. If **status** is TRUE, **code** is a negative error code. If **status** is FALSE, **code** is 0 or a warning code.
  - **source** identifies where an error occurred. The source string includes the name of the VI that produced the error, what inputs are in error, and how to eliminate the error.
- TCP Connection Refnum (dup) can be an input to another MODBUS VI, or you can close it using TCP Close Connection.vi.
- **Exception Code** is nonzero if a MODBUS error has occurred. The

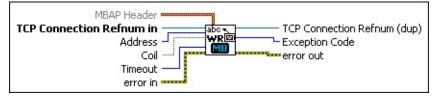
exception code mapping is as follows:

- 0 No error
- 1 Illegal function
- 2 Illegal data address
- 3 Illegal data value
- 4 Failure in associated device
- 5 Acknowledge
- 6 Busy, rejected message
- 7 NAK—Negative acknowledge

- error out contains error information. If error in indicates that an error occurred before this VI or function ran, error out contains the same error information. Otherwise, it describes the error status that this VI or function produces. Right-click the error out front panel indicator and select Explain Error from the shortcut menu for more information about the error.
  - **status** is TRUE (X) if an error occurred or FALSE (checkmark) to indicate a warning or that no error occurred.
  - **code** is the error or warning code. If **status** is TRUE, **code** is a nonzero error code. If **status** is FALSE, **code** is 0 or a warning code.
  - **source** describes the origin of the error or warning and is, in most cases, the name of the VI or function that produced the error or warning.

# MB Ethernet Master Query Write Single Coil (poly).vi

The master uses this VI to write a single coil to the slave. This corresponds to a public function code of 5 in the MODBUS protocol.



**MBAP Header** is a cluster containing the transaction ID and Unit ID. This is additional information not usually required for a MODBUS transaction.

**Transaction Identifier**—This identifier is for transaction pairing; the MODBUS server copies the request transaction identifier in the response.

**Unit Identifier**—This field is for intra-system routing purposes. It typically communicates to a MODBUS serial line slave through a gateway between an Ethernet TCP-IP network and a MODBUS serial line. The MODBUS client sets this field in the request, and the server response must return this field with the same value.

- **TCP Connection Refnum in** is a network connection refnum that uniquely identifies the TCP connection. Use TCP Open Connection.vi to open a TCP connection.
- Address is the first address location of the coil to write. This address is sometimes referred to as *offset*. Refer to your device's documentation to find the coil address mapping.

Be sure to specify the correct address in the MODBUS device configuration software for the register you want to use in LabVIEW. In most MODBUS device configuration software, you must enter a name for the register you want to use. Per MODBUS convention, the register address of the slave device is calculated by subtracting 1 from the register name that you specify in the master device configuration software. The MODBUS LabVIEW library expects register addresses, not register names, so you may need to subtract 1 from the address you defined in the MODBUS device configuration software. For example, a register name defined as 2 in a MODBUS configuration device translates to register address 1 in the Holding Registers table of the LabVIEW MODBUS library, as shown below.

```
MODBUS Holding Register Name = 2
Device
```

```
LabVIEW Holding Register Address = 1
```

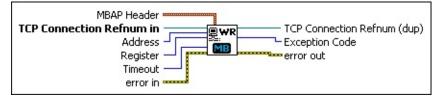
- **Coil** is the coil to write.
- **Timeout** specifies the maximum time period, in milliseconds, that the VI waits for the slave's response before throwing an error.
- error in describes error conditions that occur before this VI or function runs. The default is no error. If an error occurred before this VI or function runs, the VI or function passes the error in value to error out. If an error occurs while this VI or function runs, the VI or function runs normally and sets its own error status in error out. Use the Simple Error Handler or General Error Handler VIs to display the description of the error code. Use error in and error out to check errors and to specify execution order by wiring error out from one node to error in of the next node.
  - **status** is TRUE (X) if an error occurred before this VI or function ran or FALSE (checkmark) to indicate a warning or that no error occurred before this VI or function ran. The default is FALSE.
  - **code** is the error or warning code. The default is 0. If **status** is TRUE, **code** is a negative error code. If **status** is FALSE, **code** is 0 or a warning code.
  - **source** identifies where an error occurred. The source string includes the name of the VI that produced the error, what inputs are in error, and how to eliminate the error.
- TCP Connection Refnum (dup) can be an input to another MODBUS VI, or you can close it using TCP Close Connection.vi.
- **Exception Code** is nonzero if a MODBUS error has occurred. The exception code mapping is as follows:
  - 0 No error

- 1 Illegal function
- 2 Illegal data address
- 3 Illegal data value
- 4 Failure in associated device
- 5 Acknowledge
- 6 Busy, rejected message
- 7 NAK—Negative acknowledge

- error out contains error information. If error in indicates that an error occurred before this VI or function ran, error out contains the same error information. Otherwise, it describes the error status that this VI or function produces. Right-click the error out front panel indicator and select Explain Error from the shortcut menu for more information about the error.
  - **status** is TRUE (X) if an error occurred or FALSE (checkmark) to indicate a warning or that no error occurred.
  - **code** is the error or warning code. If **status** is TRUE, **code** is a nonzero error code. If **status** is FALSE, **code** is 0 or a warning code.
  - **source** describes the origin of the error or warning and is, in most cases, the name of the VI or function that produced the error or warning.

# MB Ethernet Master Query Write Single Register (poly).vi

The master uses this VI to write a single register to the slave. This corresponds to a public function code of 6 in the MODBUS protocol.



**MBAP Header** is a cluster containing the transaction ID and Unit ID. This is additional information not usually required for a MODBUS transaction.

**Transaction Identifier**—This identifier is for transaction pairing; the MODBUS server copies the request transaction identifier in the response.

**Unit Identifier**—This field is for intra-system routing purposes. It typically communicates to a MODBUS serial line slave through a gateway between an Ethernet TCP-IP network and a MODBUS serial line. The MODBUS client sets this field in the request, and the server response must return this field with the same value.

- **TCP Connection Refnum in** is a network connection refnum that uniquely identifies the TCP connection. Use TCP Open Connection.vi to open a TCP connection.
- Address is the address location of the register to write. This address is sometimes referred to as *offset*. Refer to your device's documentation to find the register address mapping.

Be sure to specify the correct address in the MODBUS device configuration software for the register you want to use in LabVIEW. In most MODBUS device configuration software, you must enter a name for the register you want to use. Per MODBUS convention, the register address of the slave device is calculated by subtracting 1 from the register name that you specify in the master device configuration software. The MODBUS LabVIEW library expects register addresses, not register names, so you may need to subtract 1 from the address you defined in the MODBUS device configuration software. For example, a register name defined as 2 in a MODBUS configuration device translates to register address 1 in the Holding Registers table of the LabVIEW MODBUS library, as shown below.

```
MODBUS Holding Register Name = 2
Device
```

```
LabVIEW Holding Register Address = 1
```

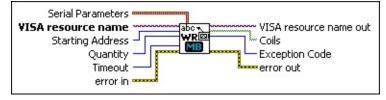
- **Register** is the register to write.
- **Timeout** specifies the maximum time period, in milliseconds, that the VI waits for the slave's response before throwing an error.
- error in describes error conditions that occur before this VI or function runs. The default is no error. If an error occurred before this VI or function runs, the VI or function passes the error in value to error out. If an error occurs while this VI or function runs, the VI or function runs normally and sets its own error status in error out. Use the Simple Error Handler or General Error Handler VIs to display the description of the error code. Use error in and error out to check errors and to specify execution order by wiring error out from one node to error in of the next node.
  - **status** is TRUE (X) if an error occurred before this VI or function ran or FALSE (checkmark) to indicate a warning or that no error occurred before this VI or function ran. The default is FALSE.
  - **code** is the error or warning code. The default is 0. If **status** is TRUE, **code** is a negative error code. If **status** is FALSE, **code** is 0 or a warning code.
  - **source** identifies where an error occurred. The source string includes the name of the VI that produced the error, what inputs are in error, and how to eliminate the error.
- TCP Connection Refnum (dup) can be an input to another MODBUS VI, or you can close it using TCP Close Connection.vi.
- **Exception Code** is nonzero if a MODBUS error has occurred. The exception code mapping is as follows:
  - 0 No error

- 1 Illegal function
- 2 Illegal data address
- 3 Illegal data value
- 4 Failure in associated device
- 5 Acknowledge
- 6 Busy, rejected message
- 7 NAK—Negative acknowledge

- error out contains error information. If error in indicates that an error occurred before this VI or function ran, error out contains the same error information. Otherwise, it describes the error status that this VI or function produces. Right-click the error out front panel indicator and select Explain Error from the shortcut menu for more information about the error.
  - **status** is TRUE (X) if an error occurred or FALSE (checkmark) to indicate a warning or that no error occurred.
  - **code** is the error or warning code. If **status** is TRUE, **code** is a nonzero error code. If **status** is FALSE, **code** is 0 or a warning code.
  - **source** describes the origin of the error or warning and is, in most cases, the name of the VI or function that produced the error or warning.

# MB Serial Master Query Read Coils (poly).vi

The master uses this VI to read the slaves' coils. This corresponds to a public function code of 1 in the MODBUS protocol.



Serial Parameters are parameters that modify the way the MODBUS frame is structured. Refer to the MODBUS specification for more information about the MODBUS Frame. The Serial Parameters in LabVIEW is a cluster containing an enum (Mode) and an integer (Slave Address).

#### Mode

**RTU** Data is represented in binary format.

#### ASCII

Data is represented in ASCII, so it is human readable.

#### **Slave Address**

This is the address of the slave you are communicating with. This property is usually valid for RS-485 networks, which can have multiple MODBUS devices connected to the same network.

**Note:** Consult your device documentation to find out how to validate/change the address of your device.

- **VISA resource name** specifies the resource to be opened. This control also specifies the session and class. For more information about VISA resource names, refer to the *NI-VISA Help*.
- **Starting Address** is the first address location of the coil to read. This address is sometimes referred to as *offset*. Refer to your device's documentation to find the coil address mapping.

Be sure to specify the correct address in the MODBUS device configuration software for the register you want to use in LabVIEW. In most MODBUS device configuration software, you must enter a name for the register you want to use. Per MODBUS convention, the register address of the slave device is calculated by subtracting 1 from the register name that you specify in the master device configuration software. The MODBUS LabVIEW library expects register addresses, not register names, so you may need to subtract 1 from the address you defined in the MODBUS device configuration software. For example, a register name defined as 2 in a MODBUS configuration device translates to register address 1 in the Holding Registers table of the LabVIEW MODBUS library, as shown below.

MODBUS Holding Register Name = 2 Device

- Quantity represents how many coils to read from the slave. The VI returns the coil at Starting Address and each following coil up to Quantity. For example, if Starting Address is 0 and Quantity is 4, the VI reads coils from the slave at address 0, 1, 2, and 3.
- **Timeout** specifies the maximum time period, in milliseconds, that the VI waits for the slave's response before throwing an error.
- error in describes error conditions that occur before this VI or function runs. The default is no error. If an error occurred before this VI or function runs, the VI or function passes the error in value to error out. If an error occurs while this VI or function runs, the VI or function runs normally and sets its own error status in error out. Use the Simple Error Handler or General Error Handler VIs to display the description of the error code. Use error in and error out to check errors and to specify execution order by wiring error out from one node to error in of the next node.
  - **status** is TRUE (X) if an error occurred before this VI or function ran or FALSE (checkmark) to indicate a warning or that no error occurred before this VI or function ran. The default is FALSE.
  - **code** is the error or warning code. The default is 0. If **status** is TRUE, **code** is a negative error code. If **status** is FALSE, **code** is 0 or a warning code.
  - **source** identifies where an error occurred. The source string includes the name of the VI that produced the error,

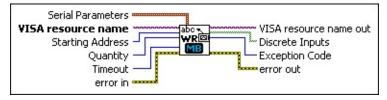
what inputs are in error, and how to eliminate the error.

- **VISA resource name out** is the resource to which a VISA session is opened and its class. The class matches that of the VISA resource name input. For more information about VISA resource names, refer to the *NI-VISA Help*.
- **Coils** represents the data read from the slave.
- **Exception Code** is nonzero if a MODBUS error has occurred. The exception code mapping is as follows:
  - 0 No error
  - 1 Illegal function
  - 2 Illegal data address
  - 3 Illegal data value
  - 4 Failure in associated device
  - 5 Acknowledge
  - 6 Busy, rejected message
  - 7 NAK—Negative acknowledge

- error out contains error information. If error in indicates that an error occurred before this VI or function ran, error out contains the same error information. Otherwise, it describes the error status that this VI or function produces. Right-click the error out front panel indicator and select Explain Error from the shortcut menu for more information about the error.
  - **status** is TRUE (X) if an error occurred or FALSE (checkmark) to indicate a warning or that no error occurred.
  - **code** is the error or warning code. If **status** is TRUE, **code** is a nonzero error code. If **status** is FALSE, **code** is 0 or a warning code.
  - **source** describes the origin of the error or warning and is, in most cases, the name of the VI or function that produced the error or warning.

# MB Serial Master Query Read Discrete Inputs (poly).vi

The master uses this VI to read the slaves' discrete inputs. This corresponds to a public function code of 2 in the MODBUS protocol.



Serial Parameters are parameters that modify the way the MODBUS frame is structured. Refer to the MODBUS specification for more information about the MODBUS Frame. The Serial Parameters in LabVIEW is a cluster containing an enum (Mode) and an integer (Slave Address).

Mode

**RTU** Data is represented in binary format.

## ASCII

Data is represented in ASCII, so it is human readable.

## **Slave Address**

This is the address of the slave you are communicating with. This property is usually valid for RS-485 networks, which can have multiple MODBUS devices connected to the same network.

**Note:** Consult your device documentation to find out how to validate/change the address of your device.

- **VISA resource name** specifies the resource to be opened. This control also specifies the session and class. For more information about VISA resource names, refer to the *NI-VISA Help*.
- **Starting Address** is the first address location of the discrete inputs to read. This address is sometimes referred to as *offset*. Refer to your device's documentation to find the discrete input address mapping.

Be sure to specify the correct address in the MODBUS device configuration software for the register you want to use in LabVIEW.

In most MODBUS device configuration software, you must enter a name for the register you want to use. Per MODBUS convention, the register address of the slave device is calculated by subtracting 1 from the register name that you specify in the master device configuration software. The MODBUS LabVIEW library expects register addresses, not register names, so you may need to subtract 1 from the address you defined in the MODBUS device configuration software. For example, a register name defined as 2 in a MODBUS configuration device translates to register address 1 in the Holding Registers table of the LabVIEW MODBUS library, as shown below.

MODBUS Holding Register Name = 2 Device

- Quantity represents how many discrete inputs to read from the slave. The VI returns the discrete input at **Starting Address** and each following discrete input up to **Quantity**. For example, if **Starting Address** is 0 and **Quantity** is 4, the VI reads discrete inputs from the slave at address 0, 1, 2, and 3.
- **Timeout** specifies the maximum time period, in milliseconds, that the VI waits for the slave's response before throwing an error.
- error in describes error conditions that occur before this VI or function runs. The default is no error. If an error occurred before this VI or function runs, the VI or function passes the error in value to error out. If an error occurs while this VI or function runs, the VI or function runs normally and sets its own error status in error out. Use the Simple Error Handler or General Error Handler VIs to display the description of the error code. Use error in and error out to check errors and to specify execution order by wiring error out from one node to error in of the next node.
  - **status** is TRUE (X) if an error occurred before this VI or function ran or FALSE (checkmark) to indicate a warning or that no error occurred before this VI or function ran. The default is FALSE.
  - **code** is the error or warning code. The default is 0. If **status** is TRUE, **code** is a negative error code. If **status** is FALSE,

code is 0 or a warning code.

- **source** identifies where an error occurred. The source string includes the name of the VI that produced the error, what inputs are in error, and how to eliminate the error.
- **VISA resource name out** is the resource to which a VISA session is opened and its class. The class matches that of the VISA resource name input. For more information about VISA resource names, refer to the *NI-VISA Help*.
- **ITE Discrete Inputs** represents the data read from the slave.
- **Exception Code** is nonzero if a MODBUS error has occurred. The exception code mapping is as follows:
  - 0 No error
  - 1 Illegal function
  - 2 Illegal data address
  - 3 Illegal data value
  - 4 Failure in associated device
  - 5 Acknowledge
  - 6 Busy, rejected message
  - 7 NAK—Negative acknowledge

For more information about these error codes, refer to your device's documentation.

- **error out** contains error information. If **error in** indicates that an error occurred before this VI or function ran, **error out** contains the same error information. Otherwise, it describes the error status that this VI or function produces. Right-click the **error out** front panel indicator and select **Explain Error** from the shortcut menu for more information about the error.
  - **status** is TRUE (X) if an error occurred or FALSE (checkmark) to indicate a warning or that no error occurred.
  - **code** is the error or warning code. If **status** is TRUE, **code** is a nonzero error code. If **status** is FALSE, **code** is 0 or a warning code.

abc

**source** describes the origin of the error or warning and is, in most cases, the name of the VI or function that produced the error or warning.

# MB Serial Master Query Read Exception Status (poly).vi

The master uses this VI to read the exception status from the slave. This corresponds to a public function code of 7 in the MODBUS protocol.



- **VISA resource name** specifies the resource to be opened. This control also specifies the session and class. For more information about VISA resource names, refer to the *NI-VISA Help*.
- **Timeout** specifies the maximum time period, in milliseconds, that the VI waits for the slave's response before throwing an error.
- error in describes error conditions that occur before this VI or function runs. The default is no error. If an error occurred before this VI or function runs, the VI or function passes the error in value to error out. If an error occurs while this VI or function runs, the VI or function runs normally and sets its own error status in error out. Use the Simple Error Handler or General Error Handler VIs to display the description of the error code. Use error in and error out to check errors and to specify execution order by wiring error out from one node to error in of the next node.
  - **status** is TRUE (X) if an error occurred before this VI or function ran or FALSE (checkmark) to indicate a warning or that no error occurred before this VI or function ran. The default is FALSE.
  - **code** is the error or warning code. The default is 0. If **status** is TRUE, **code** is a negative error code. If **status** is FALSE, **code** is 0 or a warning code.
  - **source** identifies where an error occurred. The source string includes the name of the VI that produced the error, what inputs are in error, and how to eliminate the error.
- Serial Parameters are parameters that modify the way the MODBUS frame is structured. Refer to the MODBUS specification for more information about the MODBUS Frame. The Serial

**Parameters** in LabVIEW is a cluster containing an enum (**Mode**) and an integer (**Slave Address**).

#### Mode

RTU Data is represented in binary format.

# ASCII

Data is represented in ASCII, so it is human readable.

# **Slave Address**

This is the address of the slave you are communicating with. This property is usually valid for RS-485 networks, which can have multiple MODBUS devices connected to the same network.

**Note:** Consult your device documentation to find out how to validate/change the address of your device.

- **VISA resource name out** is the resource to which a VISA session is opened and its class. The class matches that of the VISA resource name input. For more information about VISA resource names, refer to the *NI-VISA Help*.
- **Exception Code** is nonzero if a MODBUS error has occurred. The exception code mapping is as follows:
  - 0 No error
  - 1 Illegal function
  - 2 Illegal data address
  - 3 Illegal data value
  - 4 Failure in associated device
  - 5 Acknowledge
  - 6 Busy, rejected message
  - 7 NAK—Negative acknowledge

For more information about these error codes, refer to your device's documentation.

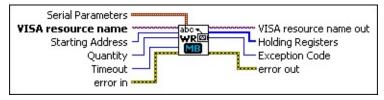
error out contains error information. If error in indicates that an error occurred before this VI or function ran, error out contains the same error information. Otherwise, it describes the error status that this VI or function produces. Right-click the error out front

panel indicator and select **Explain Error** from the shortcut menu for more information about the error.

- **status** is TRUE (X) if an error occurred or FALSE (checkmark) to indicate a warning or that no error occurred.
- **code** is the error or warning code. If **status** is TRUE, **code** is a nonzero error code. If **status** is FALSE, **code** is 0 or a warning code.
- **source** describes the origin of the error or warning and is, in most cases, the name of the VI or function that produced the error or warning.

# MB Serial Master Query Read Holding Registers (poly).vi

The master uses this VI to read the slaves' holding registers. This corresponds to a public function code of 3 in the MODBUS protocol.



Serial Parameters are parameters that modify the way the MODBUS frame is structured. Refer to the MODBUS specification for more information about the MODBUS Frame. The Serial Parameters in LabVIEW is a cluster containing an enum (Mode) and an integer (Slave Address).

Mode

**RTU** Data is represented in binary format.

## ASCII

Data is represented in ASCII, so it is human readable.

## **Slave Address**

This is the address of the slave you are communicating with. This property is usually valid for RS-485 networks, which can have multiple MODBUS devices connected to the same network.

**Note:** Consult your device documentation to find out how to validate/change the address of your device.

- **VISA resource name** specifies the resource to be opened. This control also specifies the session and class. For more information about VISA resource names, refer to the *NI-VISA Help*.
- **Starting Address** is the first address location of the holding register to read. This address is sometimes referred to as offset. Refer to your device's documentation to find the holding register address mapping.

Be sure to specify the correct address in the MODBUS device configuration software for the register you want to use in LabVIEW.

In most MODBUS device configuration software, you must enter a name for the register you want to use. Per MODBUS convention, the register address of the slave device is calculated by subtracting 1 from the register name that you specify in the master device configuration software. The MODBUS LabVIEW library expects register addresses, not register names, so you may need to subtract 1 from the address you defined in the MODBUS device configuration software. For example, a register name defined as 2 in a MODBUS configuration device translates to register address 1 in the Holding Registers table of the LabVIEW MODBUS library, as shown below.

MODBUS Holding Register Name = 2 Device

- Quantity represents how many holding registers to read from the slave. The VI returns the holding register at **Starting Address** and each following holding register up to **Quantity**. For example, if **Starting Address** is 0 and **Quantity** is 4, the VI reads holding registers from the slave at address 0, 1, 2, and 3.
- **Timeout** specifies the maximum time period, in milliseconds, that the VI waits for the slave's response before throwing an error.
- error in describes error conditions that occur before this VI or function runs. The default is no error. If an error occurred before this VI or function runs, the VI or function passes the error in value to error out. If an error occurs while this VI or function runs, the VI or function runs normally and sets its own error status in error out. Use the Simple Error Handler or General Error Handler VIs to display the description of the error code. Use error in and error out to check errors and to specify execution order by wiring error out from one node to error in of the next node.
  - **status** is TRUE (X) if an error occurred before this VI or function ran or FALSE (checkmark) to indicate a warning or that no error occurred before this VI or function ran. The default is FALSE.
  - **code** is the error or warning code. The default is 0. If **status** is TRUE, **code** is a negative error code. If **status** is FALSE,

code is 0 or a warning code.

- **source** identifies where an error occurred. The source string includes the name of the VI that produced the error, what inputs are in error, and how to eliminate the error.
- **VISA resource name out** is the resource to which a VISA session is opened and its class. The class matches that of the VISA resource name input. For more information about VISA resource names, refer to the *NI-VISA Help*.
- **Holding Registers** represents the data read from the slave (array of U16).
- **Exception Code** is nonzero if a MODBUS error has occurred. The exception code mapping is as follows:
  - 0 No error
  - 1 Illegal function
  - 2 Illegal data address
  - 3 Illegal data value
  - 4 Failure in associated device
  - 5 Acknowledge
  - 6 Busy, rejected message
  - 7 NAK—Negative acknowledge

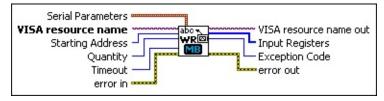
- error out contains error information. If error in indicates that an error occurred before this VI or function ran, error out contains the same error information. Otherwise, it describes the error status that this VI or function produces. Right-click the error out front panel indicator and select Explain Error from the shortcut menu for more information about the error.
  - **status** is TRUE (X) if an error occurred or FALSE (checkmark) to indicate a warning or that no error occurred.
  - **code** is the error or warning code. If **status** is TRUE, **code** is a nonzero error code. If **status** is FALSE, **code** is 0 or a warning code.

abc

**source** describes the origin of the error or warning and is, in most cases, the name of the VI or function that produced the error or warning.

# MB Serial Master Query Read Input Registers (poly).vi

The master uses this VI to read the slaves input registers. This corresponds to a public function code of 4 in the MODBUS protocol.



Serial Parameters are parameters that modify the way the MODBUS frame is structured. Refer to the MODBUS specification for more information about the MODBUS Frame. The Serial Parameters in LabVIEW is a cluster containing an enum (Mode) and an integer (Slave Address).

#### Mode

**RTU** Data is represented in binary format.

# ASCII

Data is represented in ASCII, so it is human readable.

# **Slave Address**

This is the address of the slave you are communicating with. This property is usually valid for RS-485 networks, which can have multiple MODBUS devices connected to the same network.

**Note:** Consult your device documentation to find out how to validate/change the address of your device.

- **VISA resource name** specifies the resource to be opened. This control also specifies the session and class. For more information about VISA resource names, refer to the *NI-VISA Help*.
- **Starting Address** is the first address location of the input register to read. This address is sometimes referred to as *offset*. Refer to your device's documentation to find the input register address mapping.

Be sure to specify the correct address in the MODBUS device configuration software for the register you want to use in LabVIEW.

In most MODBUS device configuration software, you must enter a name for the register you want to use. Per MODBUS convention, the register address of the slave device is calculated by subtracting 1 from the register name that you specify in the master device configuration software. The MODBUS LabVIEW library expects register addresses, not register names, so you may need to subtract 1 from the address you defined in the MODBUS device configuration software. For example, a register name defined as 2 in a MODBUS configuration device translates to register address 1 in the Holding Registers table of the LabVIEW MODBUS library, as shown below.

MODBUS Holding Register Name = 2 Device

- Quantity represents how many input registers to read from the slave. The VI returns the input register at **Starting Address** and each following input register up to **Quantity**. For example, if **Starting Address** is 0 and **Quantity** is 4, the VI reads input registers from the slave at address 0, 1, 2, and 3.
- **Timeout** specifies the maximum time period, in milliseconds, that the VI waits for the slave's response before throwing an error.
- error in describes error conditions that occur before this VI or function runs. The default is no error. If an error occurred before this VI or function runs, the VI or function passes the error in value to error out. If an error occurs while this VI or function runs, the VI or function runs normally and sets its own error status in error out. Use the Simple Error Handler or General Error Handler VIs to display the description of the error code. Use error in and error out to check errors and to specify execution order by wiring error out from one node to error in of the next node.
  - **status** is TRUE (X) if an error occurred before this VI or function ran or FALSE (checkmark) to indicate a warning or that no error occurred before this VI or function ran. The default is FALSE.
  - **code** is the error or warning code. The default is 0. If **status** is TRUE, **code** is a negative error code. If **status** is FALSE,

code is 0 or a warning code.

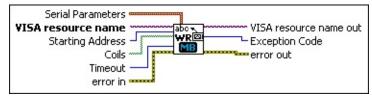
- **source** identifies where an error occurred. The source string includes the name of the VI that produced the error, what inputs are in error, and how to eliminate the error.
- **VISA resource name out** is the resource to which a VISA session is opened and its class. The class matches that of the VISA resource name input. For more information about VISA resource names, refer to the *NI-VISA Help*.
- **Input Registers** represents the data read from the slave (array of U16).
- **Exception Code** is nonzero if a MODBUS error has occurred. The exception code mapping is as follows:
  - 0 No error
  - 1 Illegal function
  - 2 Illegal data address
  - 3 Illegal data value
  - 4 Failure in associated device
  - 5 Acknowledge
  - 6 Busy, rejected message
  - 7 NAK—Negative acknowledge

- error out contains error information. If error in indicates that an error occurred before this VI or function ran, error out contains the same error information. Otherwise, it describes the error status that this VI or function produces. Right-click the error out front panel indicator and select Explain Error from the shortcut menu for more information about the error.
  - **status** is TRUE (X) if an error occurred or FALSE (checkmark) to indicate a warning or that no error occurred.
  - **code** is the error or warning code. If **status** is TRUE, **code** is a nonzero error code. If **status** is FALSE, **code** is 0 or a warning code.

**source** describes the origin of the error or warning and is, in most cases, the name of the VI or function that produced the error or warning.

# MB Serial Master Query Write Multiple Coils (poly).vi

The master uses this VI to write multiple coils to the slave. This corresponds to a public function code of 15 in the MODBUS protocol.



Serial Parameters are parameters that modify the way the MODBUS frame is structured. Refer to the MODBUS specification for more information about the MODBUS Frame. The Serial Parameters in LabVIEW is a cluster containing an enum (Mode) and an integer (Slave Address).

#### Mode

**RTU** Data is represented in binary format.

# ASCII

Data is represented in ASCII, so it is human readable.

## **Slave Address**

This is the address of the slave you are communicating with. This property is usually valid for RS-485 networks, which can have multiple MODBUS devices connected to the same network.

**Note:** Consult your device documentation to find out how to validate/change the address of your device.

- **VISA resource name** specifies the resource to be opened. This control also specifies the session and class. For more information about VISA resource names, refer to the *NI-VISA Help*.
- **Starting Address** is the address location of the first coil to write. This address is sometimes referred to as *offset*. Refer to your device's documentation to find the coil address mapping.

Be sure to specify the correct address in the MODBUS device configuration software for the register you want to use in LabVIEW. In most MODBUS device configuration software, you must enter a name for the register you want to use. Per MODBUS convention, the register address of the slave device is calculated by subtracting 1 from the register name that you specify in the master device configuration software. The MODBUS LabVIEW library expects register addresses, not register names, so you may need to subtract 1 from the address you defined in the MODBUS device configuration software. For example, a register name defined as 2 in a MODBUS configuration device translates to register address 1 in the Holding Registers table of the LabVIEW MODBUS library, as shown below.

MODBUS Holding Register Name = 2 Device

- **Coils** is the coils to write. The array length represents the number of coils that will be written.
- **Timeout** specifies the maximum time period, in milliseconds, that the VI waits for the slave's response before throwing an error.
- error in describes error conditions that occur before this VI or function runs. The default is no error. If an error occurred before this VI or function runs, the VI or function passes the error in value to error out. If an error occurs while this VI or function runs, the VI or function runs normally and sets its own error status in error out. Use the Simple Error Handler or General Error Handler VIs to display the description of the error code. Use error in and error out to check errors and to specify execution order by wiring error out from one node to error in of the next node.
  - **status** is TRUE (X) if an error occurred before this VI or function ran or FALSE (checkmark) to indicate a warning or that no error occurred before this VI or function ran. The default is FALSE.
  - **code** is the error or warning code. The default is 0. If **status** is TRUE, **code** is a negative error code. If **status** is FALSE, **code** is 0 or a warning code.
  - **source** identifies where an error occurred. The source string includes the name of the VI that produced the error, what inputs are in error, and how to eliminate the error.

**VISA resource name out** is the resource to which a VISA session is opened and its class. The class matches that of the VISA resource name input. For more information about VISA resource names, refer to the *NI-VISA Help*.

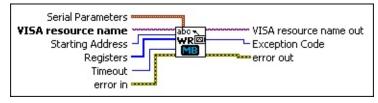
**Exception Code** is nonzero if a MODBUS error has occurred. The exception code mapping is as follows:

- 0 No error
- 1 Illegal function
- 2 Illegal data address
- 3 Illegal data value
- 4 Failure in associated device
- 5 Acknowledge
- 6 Busy, rejected message
- 7 NAK—Negative acknowledge

- error out contains error information. If error in indicates that an error occurred before this VI or function ran, error out contains the same error information. Otherwise, it describes the error status that this VI or function produces. Right-click the error out front panel indicator and select Explain Error from the shortcut menu for more information about the error.
  - **status** is TRUE (X) if an error occurred or FALSE (checkmark) to indicate a warning or that no error occurred.
  - **code** is the error or warning code. If **status** is TRUE, **code** is a nonzero error code. If **status** is FALSE, **code** is 0 or a warning code.
  - **source** describes the origin of the error or warning and is, in most cases, the name of the VI or function that produced the error or warning.

# MB Serial Master Query Write Multiple Registers (poly).vi

The master uses this VI to write multiple registers to the slave. This corresponds to a public function code of 16 in the MODBUS protocol.



Serial Parameters are parameters that modify the way the MODBUS frame is structured. Refer to the MODBUS specification for more information about the MODBUS Frame. The Serial Parameters in LabVIEW is a cluster containing an enum (Mode) and an integer (Slave Address).

#### Mode

RTU Data is represented in binary format.

## ASCII

Data is represented in ASCII, so it is human readable.

## **Slave Address**

This is the address of the slave you are communicating with. This property is usually valid for RS-485 networks, which can have multiple MODBUS devices connected to the same network.

**Note:** Consult your device documentation to find out how to validate/change the address of your device.

- **VISA resource name** specifies the resource to be opened. This control also specifies the session and class. For more information about VISA resource names, refer to the *NI-VISA Help*.
- **Starting Address** is the address location of the first register to write. This address is sometimes referred to as *offset*. Refer to your device's documentation to find the register address mapping.

Be sure to specify the correct address in the MODBUS device configuration software for the register you want to use in LabVIEW. In most MODBUS device configuration software, you must enter a name for the register you want to use. Per MODBUS convention, the register address of the slave device is calculated by subtracting 1 from the register name that you specify in the master device configuration software. The MODBUS LabVIEW library expects register addresses, not register names, so you may need to subtract 1 from the address you defined in the MODBUS device configuration software. For example, a register name defined as 2 in a MODBUS configuration device translates to register address 1 in the Holding Registers table of the LabVIEW MODBUS library, as shown below.

MODBUS Holding Register Name = 2 Device

- **Registers** are the registers to write. The array length represents how many registers will be written.
- **Timeout** specifies the maximum time period, in milliseconds, that the VI waits for the slave's response before throwing an error.
- error in describes error conditions that occur before this VI or function runs. The default is no error. If an error occurred before this VI or function runs, the VI or function passes the error in value to error out. If an error occurs while this VI or function runs, the VI or function runs normally and sets its own error status in error out. Use the Simple Error Handler or General Error Handler VIs to display the description of the error code. Use error in and error out to check errors and to specify execution order by wiring error out from one node to error in of the next node.
  - **status** is TRUE (X) if an error occurred before this VI or function ran or FALSE (checkmark) to indicate a warning or that no error occurred before this VI or function ran. The default is FALSE.
  - **code** is the error or warning code. The default is 0. If **status** is TRUE, **code** is a negative error code. If **status** is FALSE, **code** is 0 or a warning code.
  - **source** identifies where an error occurred. The source string includes the name of the VI that produced the error,

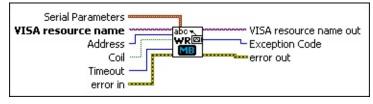
what inputs are in error, and how to eliminate the error.

- **VISA resource name out** is the resource to which a VISA session is opened and its class. The class matches that of the VISA resource name input. For more information about VISA resource names, refer to the *NI-VISA Help*.
- **Exception Code** is nonzero if a MODBUS error has occurred. The exception code mapping is as follows:
  - 0 No error
  - 1 Illegal function
  - 2 Illegal data address
  - 3 Illegal data value
  - 4 Failure in associated device
  - 5 Acknowledge
  - 6 Busy, rejected message
  - 7 NAK—Negative acknowledge

- **error out** contains error information. If **error in** indicates that an error occurred before this VI or function ran, **error out** contains the same error information. Otherwise, it describes the error status that this VI or function produces. Right-click the **error out** front panel indicator and select **Explain Error** from the shortcut menu for more information about the error.
  - **status** is TRUE (X) if an error occurred or FALSE (checkmark) to indicate a warning or that no error occurred.
  - **code** is the error or warning code. If **status** is TRUE, **code** is a nonzero error code. If **status** is FALSE, **code** is 0 or a warning code.
  - **source** describes the origin of the error or warning and is, in most cases, the name of the VI or function that produced the error or warning.

# MB Serial Master Query Write Single Coil (poly).vi

The master uses this VI to write a single coil to the slave. This corresponds to a public function code of 5 in the MODBUS protocol.



Serial Parameters are parameters that modify the way the MODBUS frame is structured. Refer to the MODBUS specification for more information about the MODBUS Frame. The Serial Parameters in LabVIEW is a cluster containing an enum (Mode) and an integer (Slave Address).

#### Mode

RTU Data is represented in binary format.

### ASCII

Data is represented in ASCII, so it is human readable.

## **Slave Address**

This is the address of the slave you are communicating with. This property is usually valid for RS-485 networks, which can have multiple MODBUS devices connected to the same network.

**Note:** Consult your device documentation to find out how to validate/change the address of your device.

- **VISA resource name** specifies the resource to be opened. This control also specifies the session and class. For more information about VISA resource names, refer to the *NI-VISA Help*.
- Address is the address location of the coil to write. This address is sometimes referred to as *offset*. Refer to your device's documentation to find the coil address mapping.

Be sure to specify the correct address in the MODBUS device configuration software for the register you want to use in LabVIEW.

In most MODBUS device configuration software, you must enter a name for the register you want to use. Per MODBUS convention, the register address of the slave device is calculated by subtracting 1 from the register name that you specify in the master device configuration software. The MODBUS LabVIEW library expects register addresses, not register names, so you may need to subtract 1 from the address you defined in the MODBUS device configuration software. For example, a register name defined as 2 in a MODBUS configuration device translates to register address 1 in the Holding Registers table of the LabVIEW MODBUS library, as shown below.

MODBUS Holding Register Name = 2 Device

- **Coil** is the coil to write.
- **Timeout** specifies the maximum time period, in milliseconds, that the VI waits for the slave's response before throwing an error.
- error in describes error conditions that occur before this VI or function runs. The default is no error. If an error occurred before this VI or function runs, the VI or function passes the error in value to error out. If an error occurs while this VI or function runs, the VI or function runs normally and sets its own error status in error out. Use the Simple Error Handler or General Error Handler VIs to display the description of the error code. Use error in and error out to check errors and to specify execution order by wiring error out from one node to error in of the next node.
  - **status** is TRUE (X) if an error occurred before this VI or function ran or FALSE (checkmark) to indicate a warning or that no error occurred before this VI or function ran. The default is FALSE.
  - **code** is the error or warning code. The default is 0. If **status** is TRUE, **code** is a negative error code. If **status** is FALSE, **code** is 0 or a warning code.
  - **source** identifies where an error occurred. The source string includes the name of the VI that produced the error, what inputs are in error, and how to eliminate the error.

**VISA resource name out** is the resource to which a VISA session is opened and its class. The class matches that of the VISA resource name input. For more information about VISA resource names, refer to the *NI-VISA Help*.

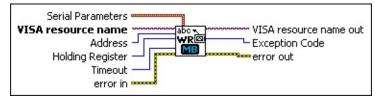
**Exception Code** is nonzero if a MODBUS error has occurred. The exception code mapping is as follows:

- 0 No error
- 1 Illegal function
- 2 Illegal data address
- 3 Illegal data value
- 4 Failure in associated device
- 5 Acknowledge
- 6 Busy, rejected message
- 7 NAK—Negative acknowledge

- **error out** contains error information. If **error in** indicates that an error occurred before this VI or function ran, **error out** contains the same error information. Otherwise, it describes the error status that this VI or function produces. Right-click the **error out** front panel indicator and select **Explain Error** from the shortcut menu for more information about the error.
  - **status** is TRUE (X) if an error occurred or FALSE (checkmark) to indicate a warning or that no error occurred.
  - **code** is the error or warning code. If **status** is TRUE, **code** is a nonzero error code. If **status** is FALSE, **code** is 0 or a warning code.
  - **source** describes the origin of the error or warning and is, in most cases, the name of the VI or function that produced the error or warning.

# MB Serial Master Query Write Single Register (poly).vi

The master uses this VI to write a single register to the slave. This corresponds to a public function code of 6 in the MODBUS protocol.



Serial Parameters are parameters that modify the way the MODBUS frame is structured. Refer to the MODBUS specification for more information about the MODBUS Frame. The Serial Parameters in LabVIEW is a cluster containing an enum (Mode) and an integer (Slave Address).

#### Mode

**RTU** Data is represented in binary format.

### ASCII

Data is represented in ASCII, so it is human readable.

## **Slave Address**

This is the address of the slave you are communicating with. This property is usually valid for RS-485 networks, which can have multiple MODBUS devices connected to the same network.

**Note:** Consult your device documentation to find out how to validate/change the address of your device.

- **VISA resource name** specifies the resource to be opened. This control also specifies the session and class. For more information about VISA resource names, refer to the *NI-VISA Help*.
- Address is the address location of the register to write. This address is sometimes referred to as *offset*. Refer to your device's documentation to find the register address mapping.

Be sure to specify the correct address in the MODBUS device configuration software for the register you want to use in LabVIEW. In most MODBUS device configuration software, you must enter a name for the register you want to use. Per MODBUS convention, the register address of the slave device is calculated by subtracting 1 from the register name that you specify in the master device configuration software. The MODBUS LabVIEW library expects register addresses, not register names, so you may need to subtract 1 from the address you defined in the MODBUS device configuration software. For example, a register name defined as 2 in a MODBUS configuration device translates to register address 1 in the Holding Registers table of the LabVIEW MODBUS library, as shown below.

MODBUS Holding Register Name = 2 Device

- Holding Register is the holding register to write.
- **Timeout** specifies the maximum time period, in milliseconds, that the VI waits for the slave's response before throwing an error.
- error in describes error conditions that occur before this VI or function runs. The default is no error. If an error occurred before this VI or function runs, the VI or function passes the error in value to error out. If an error occurs while this VI or function runs, the VI or function runs normally and sets its own error status in error out. Use the Simple Error Handler or General Error Handler VIs to display the description of the error code. Use error in and error out to check errors and to specify execution order by wiring error out from one node to error in of the next node.
  - **status** is TRUE (X) if an error occurred before this VI or function ran or FALSE (checkmark) to indicate a warning or that no error occurred before this VI or function ran. The default is FALSE.
  - **code** is the error or warning code. The default is 0. If **status** is TRUE, **code** is a negative error code. If **status** is FALSE, **code** is 0 or a warning code.
  - **source** identifies where an error occurred. The source string includes the name of the VI that produced the error, what inputs are in error, and how to eliminate the error.

- **VISA resource name out** is the resource to which a VISA session is opened and its class. The class matches that of the VISA resource name input. For more information about VISA resource names, refer to the *NI-VISA Help*.
- **Exception Code** is nonzero if a MODBUS error has occurred. The exception code mapping is as follows:
  - 0 No error
  - 1 Illegal function
  - 2 Illegal data address
  - 3 Illegal data value
  - 4 Failure in associated device
  - 5 Acknowledge
  - 6 Busy, rejected message
  - 7 NAK—Negative acknowledge

- **error out** contains error information. If **error in** indicates that an error occurred before this VI or function ran, **error out** contains the same error information. Otherwise, it describes the error status that this VI or function produces. Right-click the **error out** front panel indicator and select **Explain Error** from the shortcut menu for more information about the error.
  - **status** is TRUE (X) if an error occurred or FALSE (checkmark) to indicate a warning or that no error occurred.
  - **code** is the error or warning code. If **status** is TRUE, **code** is a nonzero error code. If **status** is FALSE, **code** is 0 or a warning code.
  - **source** describes the origin of the error or warning and is, in most cases, the name of the VI or function that produced the error or warning.

# Introduction to the Slave VIs

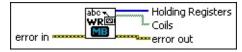
You can use the MODBUS Library VIs as masters or slaves. When used as a master, the machine that the VIs are running on sends commands or queries to the slave(s). The master reads/writes the slave's coils/discrete inputs/registers. As a slave, the machine must store this data somewhere. The slave VIs make use of arrays to store and access this data. The first VI you must use is <u>MB Slave Init (poly).vi</u>, which initializes two 65536 U16 arrays (for registers) and two 65536 Boolean arrays (for coils and discrete inputs). All other slave VIs access this memory location in different ways.

The following table describes the primary table object types and read/write types.

Primary Tables	<b>Object Type</b>	Туре
Discrete inputs	Single bit	Read only
Coils	Single bit	Read/write
Input registers	16-bit word	Read only
Holding registers	16-bit word	Read/write

# MB Slave Init (poly).vi

You must use this VI to initialize two 65536 U16 arrays (for registers) and two 65536 Boolean arrays (for coils and discrete inputs).



- error in describes error conditions that occur before this VI or function runs. The default is no error. If an error occurred before this VI or function runs, the VI or function passes the error in value to error out. If an error occurs while this VI or function runs, the VI or function runs normally and sets its own error status in error out. Use the Simple Error Handler or General Error Handler VIs to display the description of the error code. Use error in and error out to check errors and to specify execution order by wiring error out from one node to error in of the next node.
  - **status** is TRUE (X) if an error occurred before this VI or function ran or FALSE (checkmark) to indicate a warning or that no error occurred before this VI or function ran. The default is FALSE.
  - **code** is the error or warning code. The default is 0. If **status** is TRUE, **code** is a negative error code. If **status** is FALSE, **code** is 0 or a warning code.
  - **source** identifies where an error occurred. The source string includes the name of the VI that produced the error, what inputs are in error, and how to eliminate the error.
- **Holding Registers** represents an empty array for storing the holding registers.
- **Coils** represents an empty array for storing the coils.
- error out contains error information. If error in indicates that an error occurred before this VI or function ran, error out contains the same error information. Otherwise, it describes the error status that this VI or function produces. Right-click the error out front panel indicator and select Explain Error from the shortcut menu for more information about the error.

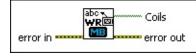
**status** is TRUE (X) if an error occurred or FALSE

(checkmark) to indicate a warning or that no error occurred.

- **code** is the error or warning code. If **status** is TRUE, **code** is a nonzero error code. If **status** is FALSE, **code** is 0 or a warning code.
- **source** describes the origin of the error or warning and is, in most cases, the name of the VI or function that produced the error or warning.

# MB Slave Read All Coils (poly).vi

Reads all coils from the slave's memory table.



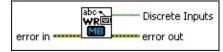
- error in describes error conditions that occur before this VI or function runs. The default is no error. If an error occurred before this VI or function runs, the VI or function passes the error in value to error out. If an error occurs while this VI or function runs, the VI or function runs normally and sets its own error status in error out. Use the Simple Error Handler or General Error Handler VIs to display the description of the error code. Use error in and error out to check errors and to specify execution order by wiring error out from one node to error in of the next node.
  - **status** is TRUE (X) if an error occurred before this VI or function ran or FALSE (checkmark) to indicate a warning or that no error occurred before this VI or function ran. The default is FALSE.
  - **code** is the error or warning code. The default is 0. If **status** is TRUE, **code** is a negative error code. If **status** is FALSE, **code** is 0 or a warning code.
  - **source** identifies where an error occurred. The source string includes the name of the VI that produced the error, what inputs are in error, and how to eliminate the error.
- **Coils** represents the entire table used for storing the coils.
- **error out** contains error information. If **error in** indicates that an error occurred before this VI or function ran, **error out** contains the same error information. Otherwise, it describes the error status that this VI or function produces. Right-click the **error out** front panel indicator and select **Explain Error** from the shortcut menu for more information about the error.
  - **status** is TRUE (X) if an error occurred or FALSE (checkmark) to indicate a warning or that no error occurred.
  - **code** is the error or warning code. If **status** is TRUE, **code** is a nonzero error code. If **status** is FALSE, **code** is 0 or a

warning code.

**source** describes the origin of the error or warning and is, in most cases, the name of the VI or function that produced the error or warning.

### MB Slave Read All Discrete Inputs (poly).vi

Reads all discrete inputs from the slave's memory table.



error in describes error conditions that occur before this VI or function runs. The default is no error. If an error occurred before this VI or function runs, the VI or function passes the error in value to error out. If an error occurs while this VI or function runs, the VI or function runs normally and sets its own error status in error out. Use the Simple Error Handler or General Error Handler VIs to display the description of the error code. Use error in and error out to check errors and to specify execution order by wiring error out from one node to error in of the next node.

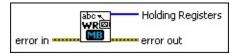
- **status** is TRUE (X) if an error occurred before this VI or function ran or FALSE (checkmark) to indicate a warning or that no error occurred before this VI or function ran. The default is FALSE.
- **code** is the error or warning code. The default is 0. If **status** is TRUE, **code** is a negative error code. If **status** is FALSE, **code** is 0 or a warning code.
- **source** identifies where an error occurred. The source string includes the name of the VI that produced the error, what inputs are in error, and how to eliminate the error.
- **Discrete Inputs** represents the entire table used for storing the discrete inputs.
- error out contains error information. If error in indicates that an error occurred before this VI or function ran, error out contains the same error information. Otherwise, it describes the error status that this VI or function produces. Right-click the error out front panel indicator and select Explain Error from the shortcut menu for more information about the error.
  - **status** is TRUE (X) if an error occurred or FALSE (checkmark) to indicate a warning or that no error occurred.
  - **code** is the error or warning code. If **status** is TRUE, **code**

is a nonzero error code. If **status** is FALSE, **code** is 0 or a warning code.

**source** describes the origin of the error or warning and is, in most cases, the name of the VI or function that produced the error or warning.

# MB Slave Read All Holding Registers (poly).vi

Reads all holding registers from the slave's memory table.



error in describes error conditions that occur before this VI or function runs. The default is no error. If an error occurred before this VI or function runs, the VI or function passes the error in value to error out. If an error occurs while this VI or function runs, the VI or function runs normally and sets its own error status in error out. Use the Simple Error Handler or General Error Handler VIs to display the description of the error code. Use error in and error out to check errors and to specify execution order by wiring error out from one node to error in of the next node.

- **status** is TRUE (X) if an error occurred before this VI or function ran or FALSE (checkmark) to indicate a warning or that no error occurred before this VI or function ran. The default is FALSE.
- **code** is the error or warning code. The default is 0. If **status** is TRUE, **code** is a negative error code. If **status** is FALSE, **code** is 0 or a warning code.
- **source** identifies where an error occurred. The source string includes the name of the VI that produced the error, what inputs are in error, and how to eliminate the error.
- **Holding Registers** represents the entire table used for storing the holding registers.
- **error out** contains error information. If **error in** indicates that an error occurred before this VI or function ran, **error out** contains the same error information. Otherwise, it describes the error status that this VI or function produces. Right-click the **error out** front panel indicator and select **Explain Error** from the shortcut menu for more information about the error.
  - **status** is TRUE (X) if an error occurred or FALSE (checkmark) to indicate a warning or that no error occurred.
  - **code** is the error or warning code. If **status** is TRUE, **code**

is a nonzero error code. If **status** is FALSE, **code** is 0 or a warning code.

**source** describes the origin of the error or warning and is, in most cases, the name of the VI or function that produced the error or warning.

# MB Slave Read All Input Registers (poly).vi

Reads all input registers from the slave's memory table.



error in describes error conditions that occur before this VI or function runs. The default is no error. If an error occurred before this VI or function runs, the VI or function passes the error in value to error out. If an error occurs while this VI or function runs, the VI or function runs normally and sets its own error status in error out. Use the Simple Error Handler or General Error Handler VIs to display the description of the error code. Use error in and error out to check errors and to specify execution order by wiring error out from one node to error in of the next node.

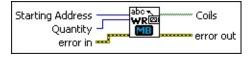
- **status** is TRUE (X) if an error occurred before this VI or function ran or FALSE (checkmark) to indicate a warning or that no error occurred before this VI or function ran. The default is FALSE.
- **code** is the error or warning code. The default is 0. If **status** is TRUE, **code** is a negative error code. If **status** is FALSE, **code** is 0 or a warning code.
- **source** identifies where an error occurred. The source string includes the name of the VI that produced the error, what inputs are in error, and how to eliminate the error.
- **Input Registers** represents the entire table used for storing the input registers.
- error out contains error information. If error in indicates that an error occurred before this VI or function ran, error out contains the same error information. Otherwise, it describes the error status that this VI or function produces. Right-click the error out front panel indicator and select Explain Error from the shortcut menu for more information about the error.
  - **status** is TRUE (X) if an error occurred or FALSE (checkmark) to indicate a warning or that no error occurred.
  - **code** is the error or warning code. If **status** is TRUE, **code**

is a nonzero error code. If **status** is FALSE, **code** is 0 or a warning code.

**source** describes the origin of the error or warning and is, in most cases, the name of the VI or function that produced the error or warning.

# MB Slave Read Coils (poly).vi

Returns the slave coils from **Starting Address** to **Starting Address** + **Quantity**.



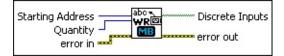
- **Starting Address** is the first address location of the coils to read. This address is sometimes referred to as *offset*.
- Quantity represents how many coils to read. The VI returns the coils at Starting Address and each following coil up to Quantity. For example, if Starting Address is 0 and Quantity is 4, the VI reads coils from addresses 0, 1, 2, and 3.
- error in describes error conditions that occur before this VI or function runs. The default is no error. If an error occurred before this VI or function runs, the VI or function passes the error in value to error out. If an error occurs while this VI or function runs, the VI or function runs normally and sets its own error status in error out. Use the Simple Error Handler or General Error Handler VIs to display the description of the error code. Use error in and error out to check errors and to specify execution order by wiring error out from one node to error in of the next node.
  - **status** is TRUE (X) if an error occurred before this VI or function ran or FALSE (checkmark) to indicate a warning or that no error occurred before this VI or function ran. The default is FALSE.
  - **code** is the error or warning code. The default is 0. If **status** is TRUE, **code** is a negative error code. If **status** is FALSE, **code** is 0 or a warning code.
  - **source** identifies where an error occurred. The source string includes the name of the VI that produced the error, what inputs are in error, and how to eliminate the error.
- **Coils** represents the data stored in the coils table.
- error out contains error information. If error in indicates that an error occurred before this VI or function ran, error out contains the same error information. Otherwise, it describes the error status

that this VI or function produces. Right-click the **error out** front panel indicator and select **Explain Error** from the shortcut menu for more information about the error.

- **status** is TRUE (X) if an error occurred or FALSE (checkmark) to indicate a warning or that no error occurred.
- **code** is the error or warning code. If **status** is TRUE, **code** is a nonzero error code. If **status** is FALSE, **code** is 0 or a warning code.
- **source** describes the origin of the error or warning and is, in most cases, the name of the VI or function that produced the error or warning.

# MB Slave Read Discrete Inputs (poly).vi

Returns the slave's discrete inputs from **Starting Address** to **Starting Address + Quantity**.



- **Starting Address** is the first address location of the discrete inputs to read. This address is sometimes referred to as *offset*.
- Quantity represents how many discrete inputs to read. The VI returns the discrete inputs at **Starting Address** and each following discrete input up to **Quantity**. For example, if **Starting Address** is 0 and **Quantity** is 4, the VI reads discrete inputs from addresses 0, 1, 2, and 3.
- error in describes error conditions that occur before this VI or function runs. The default is no error. If an error occurred before this VI or function runs, the VI or function passes the error in value to error out. If an error occurs while this VI or function runs, the VI or function runs normally and sets its own error status in error out. Use the Simple Error Handler or General Error Handler VIs to display the description of the error code. Use error in and error out to check errors and to specify execution order by wiring error out from one node to error in of the next node.
  - **status** is TRUE (X) if an error occurred before this VI or function ran or FALSE (checkmark) to indicate a warning or that no error occurred before this VI or function ran. The default is FALSE.
  - **code** is the error or warning code. The default is 0. If **status** is TRUE, **code** is a negative error code. If **status** is FALSE, **code** is 0 or a warning code.
  - **source** identifies where an error occurred. The source string includes the name of the VI that produced the error, what inputs are in error, and how to eliminate the error.
- **Discrete Inputs** represents the data stored in the discrete inputs table.
- error out contains error information. If error in indicates that an

error occurred before this VI or function ran, **error out** contains the same error information. Otherwise, it describes the error status that this VI or function produces. Right-click the **error out** front panel indicator and select **Explain Error** from the shortcut menu for more information about the error.

- **status** is TRUE (X) if an error occurred or FALSE (checkmark) to indicate a warning or that no error occurred.
- **code** is the error or warning code. If **status** is TRUE, **code** is a nonzero error code. If **status** is FALSE, **code** is 0 or a warning code.
- **source** describes the origin of the error or warning and is, in most cases, the name of the VI or function that produced the error or warning.

# MB Slave Read Holding Registers (poly).vi

Returns the slave holding registers from **Starting Address** to **Starting Address + Quantity**.



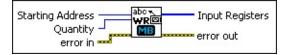
- **Starting Address** is the first address location of the holding register to read. This address is sometimes referred to as *offset*.
- Quantity represents how many holding registers to read. The VI returns the holding registers at **Starting Address** and each following holding register up to **Quantity**. For example, if **Starting Address** is 0 and **Quantity** is 4, the VI reads holding registers from addresses 0, 1, 2, and 3.
- error in describes error conditions that occur before this VI or function runs. The default is no error. If an error occurred before this VI or function runs, the VI or function passes the error in value to error out. If an error occurs while this VI or function runs, the VI or function runs normally and sets its own error status in error out. Use the Simple Error Handler or General Error Handler VIs to display the description of the error code. Use error in and error out to check errors and to specify execution order by wiring error out from one node to error in of the next node.
  - **status** is TRUE (X) if an error occurred before this VI or function ran or FALSE (checkmark) to indicate a warning or that no error occurred before this VI or function ran. The default is FALSE.
  - **code** is the error or warning code. The default is 0. If **status** is TRUE, **code** is a negative error code. If **status** is FALSE, **code** is 0 or a warning code.
  - **source** identifies where an error occurred. The source string includes the name of the VI that produced the error, what inputs are in error, and how to eliminate the error.
- **Holding Registers** represents the data read from the slave.
- error out contains error information. If error in indicates that an error occurred before this VI or function ran, error out contains the

same error information. Otherwise, it describes the error status that this VI or function produces. Right-click the **error out** front panel indicator and select **Explain Error** from the shortcut menu for more information about the error.

- **status** is TRUE (X) if an error occurred or FALSE (checkmark) to indicate a warning or that no error occurred.
- **code** is the error or warning code. If **status** is TRUE, **code** is a nonzero error code. If **status** is FALSE, **code** is 0 or a warning code.
- **Source** describes the origin of the error or warning and is, in most cases, the name of the VI or function that produced the error or warning.

# MB Slave Read Input Registers (poly).vi

Returns the slave's input registers from **Starting Address** to **Starting Address + Quantity**.



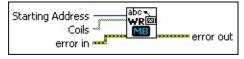
- **Starting Address** is the first address location of the input register to read. This address is sometimes referred to as *offset*.
- Quantity represents how many input registers to read. The VI returns the input registers at **Starting Address** and each following input register up to **Quantity**. For example, if **Starting Address** is 0 and **Quantity** is 4, the VI reads input registers from addresses 0, 1, 2, and 3.
- error in describes error conditions that occur before this VI or function runs. The default is no error. If an error occurred before this VI or function runs, the VI or function passes the error in value to error out. If an error occurs while this VI or function runs, the VI or function runs normally and sets its own error status in error out. Use the Simple Error Handler or General Error Handler VIs to display the description of the error code. Use error in and error out to check errors and to specify execution order by wiring error out from one node to error in of the next node.
  - **status** is TRUE (X) if an error occurred before this VI or function ran or FALSE (checkmark) to indicate a warning or that no error occurred before this VI or function ran. The default is FALSE.
  - **code** is the error or warning code. The default is 0. If **status** is TRUE, **code** is a negative error code. If **status** is FALSE, **code** is 0 or a warning code.
  - **source** identifies where an error occurred. The source string includes the name of the VI that produced the error, what inputs are in error, and how to eliminate the error.
- **Input Registers** represents the data read from the slave.
- error out contains error information. If error in indicates that an error occurred before this VI or function ran, error out contains the

same error information. Otherwise, it describes the error status that this VI or function produces. Right-click the **error out** front panel indicator and select **Explain Error** from the shortcut menu for more information about the error.

- **status** is TRUE (X) if an error occurred or FALSE (checkmark) to indicate a warning or that no error occurred.
- **code** is the error or warning code. If **status** is TRUE, **code** is a nonzero error code. If **status** is FALSE, **code** is 0 or a warning code.
- **Source** describes the origin of the error or warning and is, in most cases, the name of the VI or function that produced the error or warning.

# MB Slave Write Coils (poly).vi

Writes multiple coils to the slave memory tables.

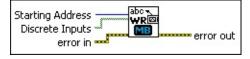


- **Starting Address** is the address location of the first coil to write. This address is sometimes referred to as *offset*.
- **Coils** to write. The array length represents the quantity.
- error in describes error conditions that occur before this VI or function runs. The default is no error. If an error occurred before this VI or function runs, the VI or function passes the error in value to error out. If an error occurs while this VI or function runs, the VI or function runs normally and sets its own error status in error out. Use the Simple Error Handler or General Error Handler VIs to display the description of the error code. Use error in and error out to check errors and to specify execution order by wiring error out from one node to error in of the next node.
  - **status** is TRUE (X) if an error occurred before this VI or function ran or FALSE (checkmark) to indicate a warning or that no error occurred before this VI or function ran. The default is FALSE.
  - **code** is the error or warning code. The default is 0. If **status** is TRUE, **code** is a negative error code. If **status** is FALSE, **code** is 0 or a warning code.
  - **source** identifies where an error occurred. The source string includes the name of the VI that produced the error, what inputs are in error, and how to eliminate the error.
- error out contains error information. If error in indicates that an error occurred before this VI or function ran, error out contains the same error information. Otherwise, it describes the error status that this VI or function produces. Right-click the error out front panel indicator and select Explain Error from the shortcut menu for more information about the error.
  - **status** is TRUE (X) if an error occurred or FALSE (checkmark) to indicate a warning or that no error occurred.

- **code** is the error or warning code. If **status** is TRUE, **code** is a nonzero error code. If **status** is FALSE, **code** is 0 or a warning code.
- **source** describes the origin of the error or warning and is, in most cases, the name of the VI or function that produced the error or warning.

# MB Slave Write Discrete Inputs (poly).vi

Writes multiple discrete inputs to the slave memory tables.



- **Starting Address** is the address location of the first discrete input to write. This address is sometimes referred to as *offset*.
- **Discrete Inputs** to write. The array length represents the quantity.

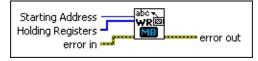
Note: Only the slave can write discrete inputs.

- error in describes error conditions that occur before this VI or function runs. The default is no error. If an error occurred before this VI or function runs, the VI or function passes the error in value to error out. If an error occurs while this VI or function runs, the VI or function runs normally and sets its own error status in error out. Use the Simple Error Handler or General Error Handler VIs to display the description of the error code. Use error in and error out to check errors and to specify execution order by wiring error out from one node to error in of the next node.
  - **status** is TRUE (X) if an error occurred before this VI or function ran or FALSE (checkmark) to indicate a warning or that no error occurred before this VI or function ran. The default is FALSE.
  - **code** is the error or warning code. The default is 0. If **status** is TRUE, **code** is a negative error code. If **status** is FALSE, **code** is 0 or a warning code.
  - **source** identifies where an error occurred. The source string includes the name of the VI that produced the error, what inputs are in error, and how to eliminate the error.
- error out contains error information. If error in indicates that an error occurred before this VI or function ran, error out contains the same error information. Otherwise, it describes the error status that this VI or function produces. Right-click the error out front panel indicator and select Explain Error from the shortcut menu for more information about the error.

- **status** is TRUE (X) if an error occurred or FALSE (checkmark) to indicate a warning or that no error occurred.
- **code** is the error or warning code. If **status** is TRUE, **code** is a nonzero error code. If **status** is FALSE, **code** is 0 or a warning code.
- **source** describes the origin of the error or warning and is, in most cases, the name of the VI or function that produced the error or warning.

# MB Slave Write Holding Registers (poly).vi

Writes multiple holding registers to the slave memory tables.



**Starting Address** is the address location of the first holding register to write. This address is sometimes referred to as *offset*.

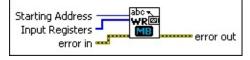
- **Holding Registers** to write. The array length represents the quantity.
- error in describes error conditions that occur before this VI or function runs. The default is no error. If an error occurred before this VI or function runs, the VI or function passes the error in value to error out. If an error occurs while this VI or function runs, the VI or function runs normally and sets its own error status in error out. Use the Simple Error Handler or General Error Handler VIs to display the description of the error code. Use error in and error out to check errors and to specify execution order by wiring error out from one node to error in of the next node.
  - **status** is TRUE (X) if an error occurred before this VI or function ran or FALSE (checkmark) to indicate a warning or that no error occurred before this VI or function ran. The default is FALSE.
  - **code** is the error or warning code. The default is 0. If **status** is TRUE, **code** is a negative error code. If **status** is FALSE, **code** is 0 or a warning code.
  - **source** identifies where an error occurred. The source string includes the name of the VI that produced the error, what inputs are in error, and how to eliminate the error.
- error out contains error information. If error in indicates that an error occurred before this VI or function ran, error out contains the same error information. Otherwise, it describes the error status that this VI or function produces. Right-click the error out front panel indicator and select Explain Error from the shortcut menu for more information about the error.
  - **status** is TRUE (X) if an error occurred or FALSE

(checkmark) to indicate a warning or that no error occurred.

- **code** is the error or warning code. If **status** is TRUE, **code** is a nonzero error code. If **status** is FALSE, **code** is 0 or a warning code.
- **source** describes the origin of the error or warning and is, in most cases, the name of the VI or function that produced the error or warning.

### MB Slave Write Input Registers (poly).vi

Writes multiple input registers to the slave memory tables.



- **Starting Address** is the address location of the first input register to write. This address is sometimes referred to as *offset*.
- **Input Registers** to write. The array length represents the quantity.

Note: Only the slave can write input registers.

- error in describes error conditions that occur before this VI or function runs. The default is no error. If an error occurred before this VI or function runs, the VI or function passes the error in value to error out. If an error occurs while this VI or function runs, the VI or function runs normally and sets its own error status in error out. Use the Simple Error Handler or General Error Handler VIs to display the description of the error code. Use error in and error out to check errors and to specify execution order by wiring error out from one node to error in of the next node.
  - **status** is TRUE (X) if an error occurred before this VI or function ran or FALSE (checkmark) to indicate a warning or that no error occurred before this VI or function ran. The default is FALSE.
  - **code** is the error or warning code. The default is 0. If **status** is TRUE, **code** is a negative error code. If **status** is FALSE, **code** is 0 or a warning code.
  - **source** identifies where an error occurred. The source string includes the name of the VI that produced the error, what inputs are in error, and how to eliminate the error.
- error out contains error information. If error in indicates that an error occurred before this VI or function ran, error out contains the same error information. Otherwise, it describes the error status that this VI or function produces. Right-click the error out front panel indicator and select Explain Error from the shortcut menu for more information about the error.

- **status** is TRUE (X) if an error occurred or FALSE (checkmark) to indicate a warning or that no error occurred.
- **code** is the error or warning code. If **status** is TRUE, **code** is a nonzero error code. If **status** is FALSE, **code** is 0 or a warning code.
- **source** describes the origin of the error or warning and is, in most cases, the name of the VI or function that produced the error or warning.

### **MB Ethernet Slave Demon.vi**

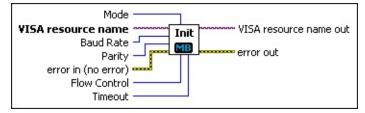
The Ethernet slave demon waits and establishes connection to Ethernet masters. It then constantly scans the TCP buffer for commands from the master. When it detects a command from the master, it updates the memory tables with data and also responds to the master's query for data. You must use the Slave API to access the data in the tables.

Port Port	# of connection	1000				
Front Panel						
2. Alles to your the day to construe to 2. Alles and the second		Memory				
name and provide more approximation and the service of the service	Modbus Slave API	Primary Tables	Object Type	Туре	Slave Demon	
Same Streeting	abc • WRE	Discrete inputs	Single bit	Read only		Serial
Levien		Coils	Single bit	Read/write		TCP/IP Master
Verchalty 35 vertice	Write Discrete Inputs 🔻	Input registers	16-bit word	Read only	MB	
		Holding registers	16-bit word	Read/write		
Part Summilian rounds, non-finit, Bin Bin Bin Bin Bin Bin Bin Bin Bin Bin						

- **Port** is the port number on which you want to listen for a connection.
- **Timeout** specifies the maximum time period, in milliseconds, that the slave waits for the master's response before throwing an error.
- **Period** represents the duration of a MODBUS cycle, or how often the slave scans the TCP buffer for commands from the masters.
- # of connections is the number of connections to TCP masters.
- **Demon Running** is true if the demon is running (listening for connections and communication with masters).

# MB Serial Init.vi

Initializes the serial port specified by **VISA resource name** to the specified settings. Wire data to the VISA resource name input to determine the polymorphic instance to use or manually select the instance.



Mode is used to select which type of MODBUS frame to use. For MB Serial Init.vi, this affects the number of data bits the serial port uses.

**RTU** Data is represented in binary format (8 data bits).

#### ASCII

Data is represented in ASCII, so it is human readable (7 data bits).

- **VISA resource name** specifies the resource to be opened. This control also specifies the session and class. For more information about VISA resource names, refer to the *NI-VISA Help*.
- **Baud Rate** is the rate of transmission. The default is 9600.
- **Parity** specifies the parity used for every frame to be transmitted or received. This input accepts the following values:
  - 0 No parity (default)
  - 1 Odd parity
  - 2 Even parity
  - 3 Mark parity
  - 4 Space parity
- error in describes error conditions that occur before this VI or function runs. The default is no error. If an error occurred before this VI or function runs, the VI or function passes the error in value to error out. If an error occurs while this VI or function runs, the VI or function runs normally and sets its own error status in error out.

Use the **Simple Error Handler** or **General Error Handler** VIs to display the description of the error code. Use **error in** and **error out** to check errors and to specify execution order by wiring **error out** from one node to **error in** of the next node.

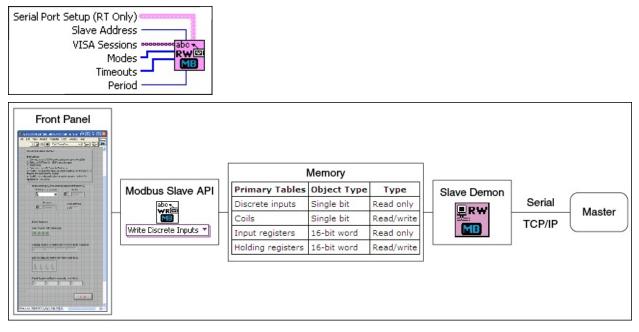
- **status** is TRUE (X) if an error occurred before this VI or function ran or FALSE (checkmark) to indicate a warning or that no error occurred before this VI or function ran. The default is FALSE.
- **code** is the error or warning code. The default is 0. If **status** is TRUE, **code** is a negative error code. If **status** is FALSE, **code** is 0 or a warning code.
- **source** identifies where an error occurred. The source string includes the name of the VI that produced the error, what inputs are in error, and how to eliminate the error.
- **Flow Control** sets the type of control the transfer mechanism uses. This input accepts the following values:
  - 0 **None (default)**—The transfer mechanism does not use flow control. Buffers on both sides of the connection are assumed to be large enough to hold all data transferred.
  - 1 **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.
  - 2 **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.
  - 3 **XON/XOFF and RTS/CTS**—The transfer mechanism uses the XON and XOFF characters and the RTS output signal and CTS input signal to perform flow control. The transfer mechanism controls input flow by sending XOFF and unasserting the RTS signal when the receive buffer is nearly full, and it controls the output flow by suspending transmission

when XOFF is received and the CTS is unasserted.

- 4 **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.
- 5 **XON/XOFF and DTR/DSR**—The transfer mechanism uses the XON and XOFF characters and the DTR output signal and DSR input signal to perform flow control. The transfer mechanism controls input flow by sending XOFF and unasserting the DTR signal when the receive buffer is nearly full, and it controls the output flow by suspending transmission when XOFF is received and the DSR signal is unasserted.
- **Timeout** specifies the maximum time period, in milliseconds, that the VI waits for the slave's response before throwing an error.
- **VISA resource name out** is the resource to which a VISA session is opened and its class. The class matches that of the VISA resource name input. For more information about VISA resource names, refer to the *NI-VISA Help*.
- **error out** contains error information. If **error in** indicates that an error occurred before this VI or function ran, **error out** contains the same error information. Otherwise, it describes the error status that this VI or function produces. Right-click the **error out** front panel indicator and select **Explain Error** from the shortcut menu for more information about the error.
  - **status** is TRUE (X) if an error occurred or FALSE (checkmark) to indicate a warning or that no error occurred.
  - **code** is the error or warning code. If **status** is TRUE, **code** is a nonzero error code. If **status** is FALSE, **code** is 0 or a warning code.
  - **Source** describes the origin of the error or warning and is, in most cases, the name of the VI or function that produced the error or warning.

### **MB Serial Slave Demon.vi**

The serial slave demon constantly scans the serial port for commands from the master. When it detects a command from the master, it updates the memory tables with data and also responds to the master's query for data. You must use the slave API to access the data in the tables.



- Serial Port Setup (RT Only) is required to configure the serial port on the RT system (if running Pharlap RT OS).
  - VISA resource name out is the resource to which a VISA session is opened and its class. The class matches that of the VISA resource name input. For more information about VISA resource names, refer to the *NI-VISA Help*.
  - **Baud Rate** is the rate of transmission. The default is 9600.
  - **Data Bits** is the number of bits in the incoming data. The value of **data bits** is between 5 and 8. The default value is 8.
  - **Parity** specifies the parity used for every frame to be transmitted or received. This input accepts the following values:
    - 0 No parity (default)
    - 1 Odd parity

- 2 Even parity
- 3 Mark parity
- 4 Space parity
- **Stop Bits** specifies the number of stop bits used to indicate the end of a frame. This input accepts the following values:
  - 10 1 stop bit
  - 15 1.5 stop bits
  - 20 2 stop bits
- **Flow Control** sets the type of control used by the transfer mechanism. This input accepts the following values:
  - 0 **None** (default)—The transfer mechanism does not use flow control. Buffers on both sides of the connection are assumed to be large enough to hold all data transferred.
  - 1 **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.
  - 2 **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.
  - 3 **XON/XOFF and RTS/CTS**—The transfer mechanism uses the XON and XOFF characters and the RTS output signal and CTS input signal to perform flow control. The transfer mechanism controls input flow by sending XOFF and unasserting the RTS signal when the receive buffer is nearly full, and it controls the output flow by suspending transmission when XOFF is received and the CTS is unasserted.

- 4 **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.
- 5 **XON/XOFF and DTR/DSR**—The transfer mechanism uses the XON and XOFF characters and the DTR output signal and DSR input signal to perform flow control. The transfer mechanism controls input flow by sending XOFF and unasserting the DTR signal when the receive buffer is nearly full, and it controls the output flow by suspending transmission when XOFF is received and the DSR signal is unasserted.
- Termination Char calls for termination of the read operation. The read operation terminates when the Termination Char is read from the serial device. 0xA is the hex equivalent of a linefeed character (\n). Change the termination char to 0xD for message strings that terminate with a carriage return (\r).
- **Enable Termination Char** prepares the serial device to recognize termination char. If TRUE (default), the port is set to recognize the termination character. If FALSE, the serial device does not recognize the termination char.
- Timeout specifies the maximum time period, in milliseconds, that the VI waits for the slave's response before throwing an error.
- Slave Address represents the address of the slave on the MODBUS network. The masters use this address to communicate with the correct slave.
- VISA resource names specifies a array of all the resources to be opened and used to connect to masters. For example, you can use two serial ports to connect a slave to multiple masters. Because this is an array of strings, you can use a VISA property node to extract the VISA resource name from a VISA Session control. Refer to MB Serial Slave Example.vi for an example.

Modes selects which type of MODBUS frame to use. This is an array of clusters. Each array location corresponds to the mode setting for the corresponding VISA session in the VISA resource names array.

#### RTU

Data is represented in binary format (8 data bits).

#### ASCII

Data is represented in ASCII, so it is human readable (7 data bits).

- Timeouts specifies the maximum time period, in milliseconds, that the slave waits for the master's response before throwing an error. Each location in the array represents the setting for the corresponding VISA resource from the VISA resource names array.
- **Period** represents the duration of a MODBUS cycle, or how often the slave scans the serial ports.

### **Important Information**

Warranty Copyright Trademarks Patents Warning Regarding Use of NI Products

### Warranty

The media on which you receive National Instruments software are warranted not to fail to execute programming instructions, due to defects in materials and workmanship, for a period of 90 days from date of shipment, as evidenced by receipts or other documentation. National Instruments will, at its option, repair or replace software media that do not execute programming instructions if National Instruments receives notice of such defects during the warranty period. National Instruments does not warrant that the operation of the software shall be uninterrupted or error free.

A Return Material Authorization (RMA) number must be obtained from the factory and clearly marked on the outside of the package before any equipment will be accepted for warranty work. National Instruments will pay the shipping costs of returning to the owner parts which are covered by warranty.

National Instruments believes that the information in this document is accurate. The document has been carefully reviewed for technical accuracy. In the event that technical or typographical errors exist, National Instruments reserves the right to make changes to subsequent editions of this document without prior notice to holders of this edition. The reader should consult National Instruments if errors are suspected. In no event shall National Instruments be liable for any damages arising out of or related to this document or the information contained in it.

EXCEPT AS SPECIFIED HEREIN, NATIONAL INSTRUMENTS MAKES NO WARRANTIES, EXPRESS OR IMPLIED, AND SPECIFICALLY DISCLAIMS ANY WARRANTY OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. CUSTOMER'S RIGHT TO RECOVER DAMAGES CAUSED BY FAULT OR NEGLIGENCE ON THE PART OF NATIONAL INSTRUMENTS SHALL BE LIMITED TO THE AMOUNT THERETOFORE PAID BY THE CUSTOMER. NATIONAL INSTRUMENTS WILL NOT BE LIABLE FOR DAMAGES RESULTING FROM LOSS OF DATA, PROFITS, USE OF PRODUCTS, OR INCIDENTAL OR CONSEQUENTIAL DAMAGES, EVEN IF ADVISED OF THE POSSIBILITY THEREOF. This limitation of the liability of National Instruments will apply regardless of the form of action, whether in contract or tort, including negligence. Any action against National Instruments must be brought within one year after the cause of action accrues. National Instruments shall not be liable for any delay in performance due to causes beyond its reasonable control. The warranty provided herein does not cover damages, defects, malfunctions, or service failures caused by owner's failure to follow the National Instruments installation, operation, or maintenance instructions; owner's modification of the product; owner's abuse, misuse, or negligent acts; and power failure or surges, fire, flood, accident, actions of third parties, or other events outside reasonable control.

# Copyright

Under the copyright laws, this publication may not be reproduced or transmitted in any form, electronic or mechanical, including photocopying, recording, storing in an information retrieval system, or translating, in whole or in part, without the prior written consent of National Instruments Corporation.

National Instruments respects the intellectual property of others, and we ask our users to do the same. NI software is protected by copyright and other intellectual property laws. Where NI software may be used to reproduce software or other materials belonging to others, you may use NI software only to reproduce materials that you may reproduce in accordance with the terms of any applicable license or other legal restriction.

### Trademarks

National Instruments, NI, ni.com, and LabVIEW are trademarks of National Instruments Corporation. Refer to the *Terms of Use* section on ni.com/legal for more information about <u>National Instruments trademarks</u>.

FireWire® is the registered trademark of Apple Inc.

Handle Graphics®, MATLAB®, Real-Time Workshop®, Simulink®, Stateflow®, and xPC TargetBox® are registered trademarks, and TargetBox™ and Target Language Compiler™ are trademarks of The MathWorks, Inc.

Tektronix® and Tek are registered trademarks of Tektronix, Inc.

The Bluetooth  $\ensuremath{\mathbb{R}}$  word mark is a registered trademark owned by the Bluetooth SIG, Inc.

Other product and company names mentioned herein are trademarks or trade names of their respective companies.

Members of the National Instruments Alliance Partner Program are business entities independent from National Instruments and have no agency, partnership, or joint-venture relationship with National Instruments.

### Patents

For patents covering the National Instruments products/technology, refer to the appropriate location: **Help»Patents** in your software, the patents.txt file on your media, or the <u>National Instruments Patent Notice</u> at ni.com/patents.

### WARNING REGARDING USE OF NATIONAL INSTRUMENTS PRODUCTS

(1) NATIONAL INSTRUMENTS PRODUCTS ARE NOT DESIGNED WITH COMPONENTS AND TESTING FOR A LEVEL OF RELIABILITY SUITABLE FOR USE IN OR IN CONNECTION WITH SURGICAL IMPLANTS OR AS CRITICAL COMPONENTS IN ANY LIFE SUPPORT SYSTEMS WHOSE FAILURE TO PERFORM CAN REASONABLY BE EXPECTED TO CAUSE SIGNIFICANT INJURY TO A HUMAN.

(2) IN ANY APPLICATION, INCLUDING THE ABOVE, RELIABILITY OF OPERATION OF THE SOFTWARE PRODUCTS CAN BE IMPAIRED BY ADVERSE FACTORS, INCLUDING BUT NOT LIMITED TO FLUCTUATIONS IN ELECTRICAL POWER SUPPLY, COMPUTER HARDWARE MALFUNCTIONS, COMPUTER OPERATING SYSTEM SOFTWARE FITNESS, FITNESS OF COMPILERS AND DEVELOPMENT SOFTWARE USED TO DEVELOP AN APPLICATION, INSTALLATION ERRORS. SOFTWARE AND HARDWARE COMPATIBILITY PROBLEMS, MALFUNCTIONS OR FAILURES OF ELECTRONIC MONITORING OR CONTROL DEVICES, TRANSIENT FAILURES OF ELECTRONIC SYSTEMS (HARDWARE AND/OR SOFTWARE), UNANTICIPATED USES OR MISUSES, OR ERRORS ON THE PART OF THE USER OR APPLICATIONS DESIGNER (ADVERSE FACTORS SUCH AS THESE ARE HEREAFTER COLLECTIVELY TERMED "SYSTEM FAILURES"). ANY APPLICATION WHERE A SYSTEM FAILURE WOULD CREATE A RISK OF HARM TO PROPERTY OR PERSONS (INCLUDING THE RISK OF BODILY INJURY AND DEATH) SHOULD NOT BE RELIANT SOLELY UPON ONE FORM OF ELECTRONIC SYSTEM DUE TO THE RISK OF SYSTEM FAILURE. TO AVOID DAMAGE. INJURY. OR DEATH. THE USER OR APPLICATION DESIGNER MUST TAKE REASONABLY PRUDENT STEPS TO PROTECT AGAINST SYSTEM FAILURES. INCLUDING BUT NOT LIMITED TO BACK-UP OR SHUT DOWN MECHANISMS. BECAUSE EACH END-USER SYSTEM IS CUSTOMIZED AND DIFFERS FROM NATIONAL INSTRUMENTS' TESTING PLATFORMS AND BECAUSE A USER OR APPLICATION DESIGNER MAY USE NATIONAL INSTRUMENTS PRODUCTS IN COMBINATION WITH OTHER PRODUCTS IN A MANNER NOT EVALUATED OR CONTEMPLATED BY NATIONAL INSTRUMENTS, THE USER OR

APPLICATION DESIGNER IS ULTIMATELY RESPONSIBLE FOR VERIFYING AND VALIDATING THE SUITABILITY OF NATIONAL INSTRUMENTS PRODUCTS WHENEVER NATIONAL INSTRUMENTS PRODUCTS ARE INCORPORATED IN A SYSTEM OR APPLICATION, INCLUDING, WITHOUT LIMITATION, THE APPROPRIATE DESIGN, PROCESS AND SAFETY LEVEL OF SUCH SYSTEM OR APPLICATION.

### **Technical Support and Professional Services**

Visit the following sections of the award-winning National Instruments Web site at ni.com for technical support and professional services:

- <u>Support</u>—Technical support at ni.com/support includes the following resources:
  - Self-Help Resources—For answers and solutions, visit ni.com/support for software drivers and updates, a searchable KnowledgeBase, product manuals, step-bystep troubleshooting wizards, thousands of example programs, tutorials, application notes, instrument drivers, and so on. Registered users also receive access to the NI Discussion Forums at ni.com/forums. NI Applications Engineers make sure every question submitted online receives an answer.
  - Standard Service Program Membership—This program entitles members to direct access to NI Applications Engineers via phone and email for one-toone technical support, as well as exclusive access to on demand training modules via the <u>Services Resource</u> <u>Center</u>. NI offers complementary membership for a full year after purchase, after which you may renew to continue your benefits.

For information about other <u>technical support options</u> in your area, visit ni.com/services or <u>contact</u> your local office at ni.com/contact.

- <u>Training and Certification</u>—Visit ni.com/training for self-paced training, eLearning virtual classrooms, interactive CDs, and Certification program information. You also can register for instructor-led, hands-on courses at locations around the world.
- <u>System Integration</u>—If you have time constraints, limited inhouse technical resources, or other project challenges, National Instruments Alliance Partner members can help. To learn more, call your local NI office or visit ni.com/alliance.

If you searched ni.com and could not find the answers you need, contact your <u>local office</u> or NI corporate headquarters. You also can visit the <u>Worldwide Offices</u> section of ni.com/niglobal to access the branch office Web sites, which provide up-to-date contact information, support phone numbers, email addresses, and current events.



**Note** Modbus LabVIEW library support is by e-mail only; phone support is not available.

### **Branch Offices**

Office	Telephone Number				
Australia	1800 300 800				
Austria	43 662 457990-0				
Belgium	32 (0) 2 757 0020				
Brazil	55 11 3262 3599				
Canada	800 433 3488				
China	86 21 5050 9800				
Czech Republic	420 224 235 774				
Denmark	45 45 76 26 00				
Finland	358 (0) 9 725 72511				
France	33 (0) 1 57 66 24 24				
Germany	49 89 7413130				
India	91 80 41190000				
Israel	972 0 3 6393737				
Italy	39 02 41309277				
Japan	0120-527196 / 81 3 5472 2970				
Korea	82 02 3451 3400				
Lebanon	961 (0) 1 33 28 28				
Malaysia	1800 887710				
Mexico	01 800 010 0793				
Netherlands	31 (0) 348 433 466				
New Zealand	0800 553 322				
Norway	47 (0) 66 90 76 60				
Poland	48 22 328 90 10				
Portugal	351 210 311 210				
Russia	7 495 783 6851				
Singapore	1800 226 5886				
Slovenia	386 3 425 42 00				

South Africa	27 0 11 805 8197			
Spain	34 91 640 0085			
Sweden	46 (0) 8 587 895 00			
Switzerland	41 56 2005151			
Taiwan	886 02 2377 2222			
Thailand	662 278 6777			
Turkey	90 212 279 3031			
United Kingdom	44 (0) 1635 523545			
United States (Corporate)	512 683 0100			