Application Note

Using Qlarity-based Terminals as a Modbus™ Master

Introduction

When using the Modbus protocol, it is common that a terminal, such as a QTERM-G70 act as the Modbus Master, and the controller or other device act as the Modbus Slave. This application note will provide instructions for the most common case and instruct you how to use the Qlarity Modbus library to communicate with a Modbus Slave.

It is helpful to have a basic knowledge of Qlarity Foundry™. A free copy of Qlarity Foundry, Qlarity™ tutorials, documentation and other resources are available at www.qlarity.com.

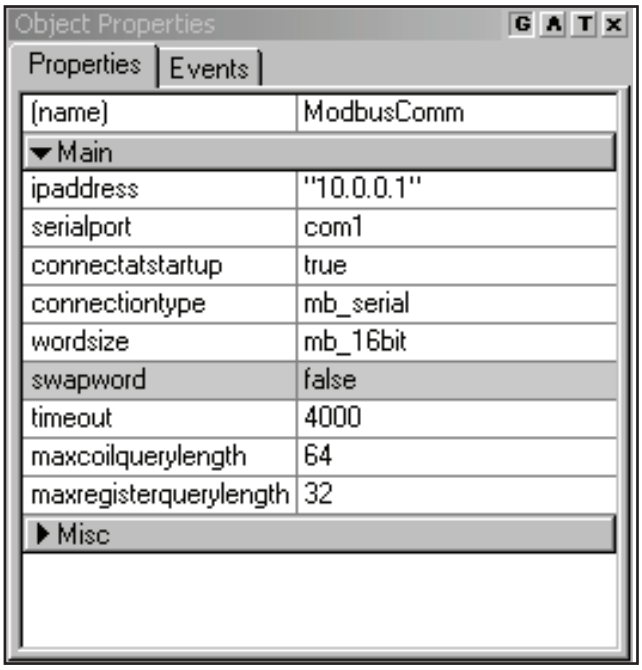
The ModbusComm Object

First, we will launch Qlarity Foundry and start a new workspace. When the "Select the Type of Workspace to Create" window appears, select the "Modbus™ serial RTU mode and Open Modbus™ /TCP" option. This will automatically include the library ModbusV2.qlib and create an initial ModbusComm object.

Select the ModbusComm object in the Object Tree and notice some of the properties of this object in the Object Properties window in the lower right of the screen. Some commonly used properties for this object are ipaddress, serialport, connectiontype, and swapword. If you are going to be communicating on a serial connection, make sure that the connectiontype is set to mb_serial and the serialport is set to the port that you will be using (e.g. COM1). If your are usint Modbus TCP, make sure that the connectiontype is set to mb_tcp and the ipaddress is set. See example at right.

The MBRegisterV2 Object

Create an instance of the MBRegisterV2 object and adjust its properties. Verify that the plc, registertype, register and vartype properties are configured to read a value from your modbus slave.



The plc property is typically set to "1" unless you have multiple slaves on a 485 network.

The registertype property can be set to mb_holdingregisters which uses readable and writeable addresses or to mb_inputregisters which use read only.

The register property specifies which address to read the data from and typically ranges between 1-10000.

The vartype property specifies how the bits are to be converted after a result is received from a query.

Test Communications with the Slave

After configuring the MBRegisterV2 object, we should try communicating with the Modbus Slave using the Simulation View in Qlarity Foundry. First, you will need to configure the communication settings in the Tool menu in Qlarity Foundry.

From the menu, select Tool | Settings | Simulation tab. If you see the message "Communication

Timeout on Reply", verify cables, communication settings and ModbusComm properties are correct. If you are receive unexpected values for 32 bit integers or floats try toggling the swapword property of the ModbusComm object.

If you are downloading the application to a Qlarity terminal, use the Power On Setup menu to configure the serial ports or IP Address. See the "Qlarity-based Terminal Hardware Manual" for information on Power On Setup.

The MBVirtualRegister Object

An MBVirtualRegister object can be used to read a register and execute different actions in your program based on its value. This is a non-drawable object with an event called ValueChanged() that is called whenever a new value is read.

Below is a screen capture from the mbexample1.qly file that can be downloaded from our website www.qlarity.com.

Changing the Value of a Register

One simple way to change the value of a register is using the MBRegisterV2 object. Find its property usersetable and change to true. Now that this feature is enabled you can click on the object, or use the enter key (if your using a keypad) to enable an editing object such as a numberkeypad.

Example:

```
'in ButtonV2's Click event
func click()
    mbregister_1.value = "5"
    mbregister_1.registervalue = 5.0
endfunc
```