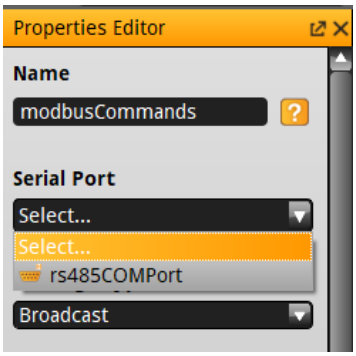
Getting Started

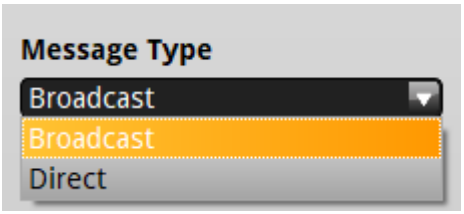
Modbus communication properties and modes can be set through the **rs485COMPort** hardware element. Reading and writing data to another device through Modbus requires the use of the **Modbus Commands** function element. To connect the two elements, select the *Serial Port* dropdown at the top of the **Modbus Commands** properties editor and choose the hardware element that you created.



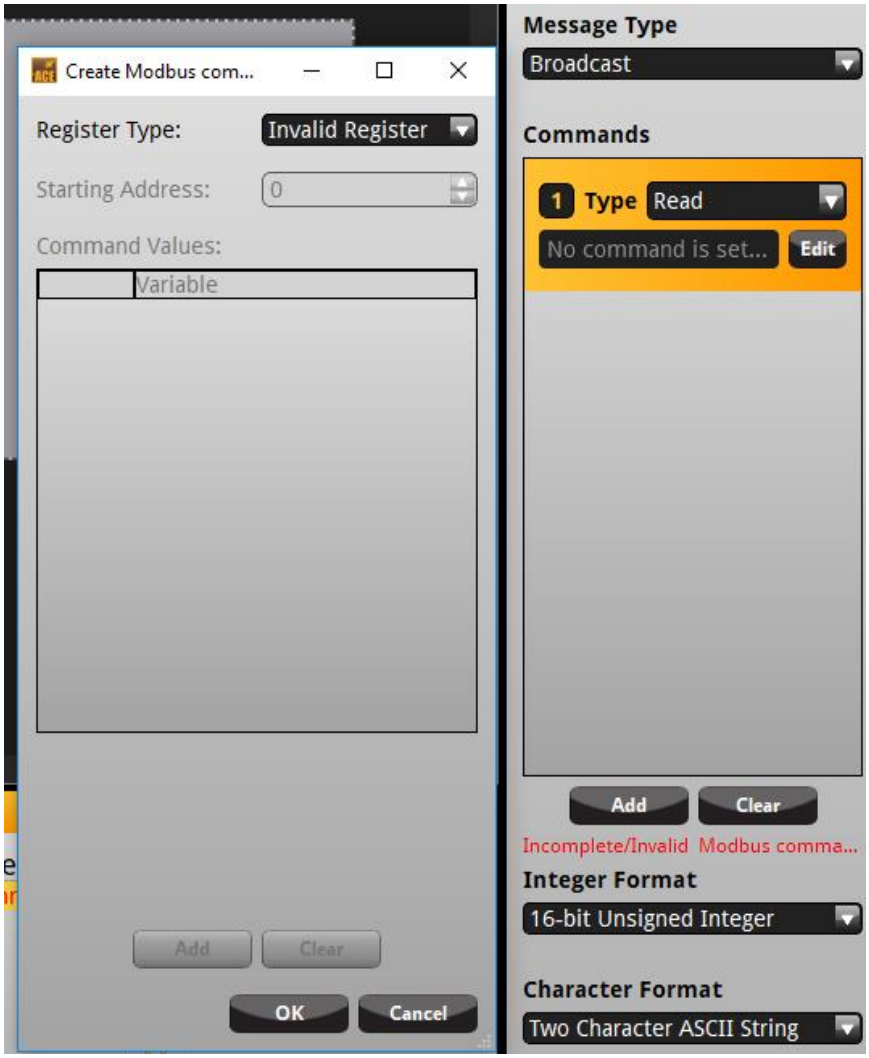
In the **rs485COMPort** properties, make sure the baud rate and byte order format are set correctly to what your device uses, as these are a common source of errors. Also make sure that the baud rate (and other settings) on the hardware element do not vary between screens, as this may cause the PanelPilotACE to crash when running the project.

Reading/Writing Data

PanelPilotACE Design Studio allows the use of both direct and broadcast messages by changing the *Message Type* property in the **Modbus Commands** element. After choosing the correct message type (and slave address if using direct), you can then add a sequence of read/write commands to be performed.



Click the 'Add' button on the *Commands* property and select a *Type*. Then, click the 'Edit' button to bring up the command dialog, and select the register that you would like to read or write to. If you find that all of them are greyed out, you may have forgotten to select the *Type*.



You may then select a starting address. Any command values added will begin at the chosen starting address, and then increment by the appropriate index. The amount of data pulled and the increment will vary according to the *Integer Format* setting, so if you need to read/write both 16- and 32-bit values, you will have to separate the read/writes into multiple **Modbus Command** elements. The address offset is already applied by the element (i.e. 40001 for holding registers), so there is no need to account for it.

When reading values, you may choose to read it into a project variable or element property that matches the type of the value pulled. For example, if the command pulls an integer, you could set the size of a text box but not the text itself. If you are looking to display it on a page, it may be best to do so by assigning the value to a project variable and then assigning the variable to a text box through the **Action (Set Rule)** function element, using a **String Converter** if necessary. Writing values works the same way, except you can also write constants in addition to variables and element properties. If you want to use the text value of a text box, you may need to use a **Maths Builder** to convert the string value to a number.

Floating Point Values

If your device handles floating points by multiplying a power of 10 to the value to convert the float into an integer: save the value to a variable and use the **Maths Builder** to scale it for reading/writing as well as for type conversion. **Maths Builder** handles conversion through the *Decimal Places* property; setting the decimal place to 0 converts the value to an integer.

Otherwise, you may use 32-bit Single Precision Floating Point format as the *Floating Format* (currently, the only option available), and set the byte order to match the target device format. Design Studio will auto check the data type set in the **Modbus Command's** Command Values.

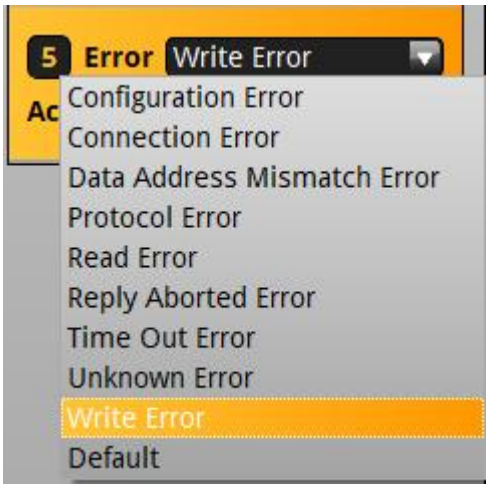
Control Flow

Many function elements allow you to determine when and under what conditions the Modbus commands should be executed. This will involve the *Action* or *On Clicked/Checked* property which determines what function to execute after the first function either completes or reaches a specific state. These are just some of the ways you can alter how the commands get executed:

- Running the commands immediately: use a **Property Trigger**, with the *Element* set to the current screen and the *Property* set to "Added to Scene".
- On a delay/on an interval: **Timer** element, with desired *Start Delay* and/or *Interval* values.
- After certain conditions are met: **Logic Builder** element. Action gets executed when the expression evaluates to true.
- One after another: on the first **Modbus Command**, look for the *Post Action* property and set the next command to be executed there.

Handling Errors

Errors that occur during communication, such as configuration issues, address mismatching, connection issues, and timeout can be addressed by the **Communication Error Handler** element.



Create one and link it to the **Modbus Command** element you want to error check by setting the Command's *Error Handler* property.