PVBC - Protocol and Device Configuration

# Overview

PV Bean Counter (PVBC) now supports configuration of new devices using parameter files. These parameter files describe:

* Low level device "Conversation" protocols. These provide message assembly and framing for outgoing messages. For incoming messages they provide message identification and component disassembly
* Higher level "Device" definitions. These provide mappings between a physical device accessible by a defined protocol, and a standard PVBC device model (or type). The only standard model currently available is the "Inverter" model

# Protocol Definitions

A "Protocol " definition contains one or more "Conversation" definitions.

A Conversation definition contains one or more "Message" definitions.

A Message definition contains a number of "Elements".

This section will explain how to describe a new protocol by explaining two protocols currently used by PVBC.

In the version of PVBC current when this document was published, all protocol definitions are stored in the file named DeviceManagement\_v02.xml. This file is overwritten whenever a new version of PVBC is installed. Modifications to this file must be maintained in a copy outside the PVBC software installation directory to avoid loss during PVBC installation.

## The Query / Response Protocol

This is a protocol that can be used to conduct general purpose conversations with a. It sends a simple (single element) command to the device and receives a simple response from the device. The response can contain from one to four separate elements but the number of elements in the response must be known for each request before the request is issued.

This example is used for non-Modbus communication with Xantrex inverters. It contains no Xantrex specific detail and can be used by other devices that use a similar request / response pattern.

The protocol definition starts with the following xml elements:

<protocol>

<name value="QueryResponse\_Generic" />"

<type value="QueryResponse" />

<protocol> commences the definition of a new protocol

<name> provides the unique protocol name. This is referenced from the higher level device definitions

<type> identifies the PVBC internal protocol handler that can correctly implement this protocol definition. *"QueryResponse"* is the general purpose protocol handler. "Modbus" is a protocol handler that implements Modbus specific algorithms that simplify access to Modbus devices

### The QueryResponse Conversation

Following is the first of five conversations defined within this Protocol:

<conversation>

<name value="QueryResponse" />

<message>

<type value="Send" />

<element type="STRING" name="QueryString" />

<element type="BYTE" name="EndOfMessage" />

</message>

<message>

<type value="Extract" />

<element type="DYNAMICBYTE" name="Response" />

<element type="BYTE" name="EndOfMessage" />

</message>

</conversation>

This is a simple conversation that sends a request to a device and receives a response containing a single variable element.

The element named *QueryString* will be assigned the value of the request that will be sent to the device.

The element named *EndOfMessage* will contain the byte sequence that marks the end of a message.

The element named *Response* will be loaded with the response data received from the device up to but excluding the end of message marker.

<name> provides the Conversation Name. This is referenced from higher level Device definitions.

<message> describes a message in the conversation. The messages are executed in the order in which they are defined. If a message fails due to some data content of communication issue, the conversation is terminated

<type> identifies how the message is handled in the conversation.

"Send" causes the message to be constructed from the defined elements and sent to the device

"Read" causes the bytes from the input stream to be read and matched with the current content of the elements of the message. The read succeeds if all elements in the message match the bytes read from the input stream

"Extract" causes bytes from the input stream to be copied into the elements of the message. Where the message elements have a predefined size, the bytes are simply copied into the elements based on the element size and order. Normally all elements in an "Extract" message must have fixed sizes and must be variables (not literals). There is one exception that applies when the first element in the message has a type of "DYNAMICBYTE". See below for details of DYNAMICBYTE behaviour

"Find" causes bytes in the input stream to be discarded until bytes are located that match the pattern defined by the current values of the elements in the message definition

<element> defines one component of the message. These elements can be named. Named elements can be referenced from higher level device definitions. Elements are defined by their attributes:

name= provides an element name that allows this element to be bound to "Registers" defined in the higher level device definitions

type= defines the data type of the element. This affects the way data is transformed as it is loaded into and retrieved from the element. It also affects the way the element content is represented "on the wire":

"STRING" - Element content is treated as ASCII characters, one character to each byte "On The Wire". Numeric data assigned to this element type is converted to string format before being assigned

"BYTE" - Element content is treated as 8bit bytes. String data assigned to this element type is treated as hexadecimal if the string starts with "0x" or "0X". Otherwise string data is loaded as for "String" described above

"DYNAMICBYTE" - Used only in Extract messages. This element has no defined size. It will accept all data in the input stream up to the pattern defined by other elements that follow the this element in the message definition. This allows the bounds of a variable length response to be detected

### The QueryResponseX2 Conversation

Following is the second conversation defined within this Protocol:

<conversation>

<name value="QueryResponseX2" />

<message>

<type value="Send" />

<element type="STRING" name="QueryString" />

<element type="BYTE" name="EndOfMessage" />

</message>

<message>

<type value ="Read" />

<element type="BYTE" name="Marker1" />

</message>

<message>

<type value="Extract" />

<element type="DYNAMICBYTE" name="Response1" />

<element type="BYTE" name="Marker2" />

</message>

<message>

<type value="Extract" />

<element type="DYNAMICBYTE" name="Response2" />

<element type="BYTE" name="Marker3" />

<element type="BYTE" name="EndOfMessage" />

</message>

</conversation>

</protocol>

This conversation sends a request to a device and receives a response containing a two variable elements. The elements named *Response1* and *Response2* will contain the variables from the device response after execution of this conversation.

The elements named *Marker1, Marker2* and *Marker3* are loaded before the conversation is executed. They will contain the byte arrays that frame and delimit the variable length response values.

### The QueryResponseX3 and QueryResponseX3 Conversations

Following are the third and fourth conversation defined within this Protocol:

<conversation>

<name value="QueryResponseX3" />

<message>

<type value="Send" />

<element type="STRING" name="QueryString" />

<element type="BYTE" name="EndOfMessage" />

</message>

<message>

<type value ="Read" />

<element type="BYTE" name="Marker1" />

</message>

<message>

<type value="Extract" />

<element type="DYNAMICBYTE" name="Response1" />

<element type="BYTE" name="Marker2" />

</message>

<message>

<type value="Extract" />

<element type="DYNAMICBYTE" name="Response2" />

<element type="BYTE" name="Marker3" />

</message>

<message>

<type value="Extract" />

<element type="DYNAMICBYTE" name="Response3" />

<element type="BYTE" name="Marker4" />

<element type="BYTE" name="EndOfMessage" />

</message>

</conversation>

<conversation>

<name value="QueryResponseX4" />

<message>

<type value="Send" />

<element type="STRING" name="QueryString" />

<element type="BYTE" name="EndOfMessage" />

</message>

<message>

<type value ="Read" />

<element type="BYTE" name="Marker1" />

</message>

<message>

<type value="Extract" />

<element type="DYNAMICBYTE" name="Response1" />

<element type="BYTE" name="Marker2" />

</message>

<message>

<type value="Extract" />

<element type="DYNAMICBYTE" name="Response2" />

<element type="BYTE" name="Marker3" />

</message>

<message>

<type value="Extract" />

<element type="DYNAMICBYTE" name="Response3" />

<element type="BYTE" name="Marker4" />

</message>

<message>

<type value="Extract" />

<element type="DYNAMICBYTE" name="Response4" />

<element type="BYTE" name="Marker5" />

<element type="BYTE" name="EndOfMessage" />

</message>

</conversation>

These conversations are similar to *"RequestResponseX2"* . They are used to send requests that provide 3 or 4 (respectively) variable length components in a single device response. Clearly this pattern can be extended as required to handle larger numbers of response components.

## The Modbus Protocol

The Modbus protocol could be implemented using the generic *QueryResponse* protocol defined above. This Modbus specific protocol definition allows the Modbus algorithms available within PVBC to be leveraged to simplify the definition of Modbus devices.

The protocol definition starts with the following xml elements:

<protocol>

<name value="Modbus" />

<type value="Modbus" />

<checksum value="ModbusCheckSum16" />

<checksumendian16bit value="Little" />

<endian16bit value="Big" />

<endian32bit value="Big" />

<protocol> commences the definition of a new protocol

<name> provides the unique protocol name. This is referenced from the higher level device definitions

<type> identifies the PVBC internal protocol handler that can correctly implement this protocol definition. "Modbus" is a protocol handler that implements Modbus specific algorithms that simplify access to Modbus devices

<checksum> provides the name of the checksum calculation algorithm that will be used for all *CheckSum16* message elements

<checksumendian16bit> specifies the byte order to be used in this protocol for checksum values:

"Big" - The high-order byte is sent "On The Wire" before the low-order byte

"Little" - The low-order byte is sent "On The Wire" before the high-order byte. Note that the Modbus protocol is unusual in that it uses a different byte order for checksum values to that used for 16bit numbers

<endian16bit> specifies the byte order to be used in this protocol for 16bit numbers:

"Big" - The high-order byte is sent "On The Wire" before the low-order byte

"Little" - The low-order byte is sent "On The Wire" before the high-order byte

<endian32bit> specifies the byte order to be used in this protocol for 32bit numbers:

"Big" - The high-order 16bit word is send is sent "On The Wire" before the low-order 16bit word. Within each 16bit word, the high-order byte is sent "On The Wire" before the low-order byte

"Little" - The low-order 16bit word is send is sent "On The Wire" before the high-order 16bit word. Within each 16bit word, the low-order byte is sent "On The Wire" before the high-order byte

"BigLittle" - The low-order 16bit word is sent "On The Wire" before the high-order 16bit word. Within each 16bit word, the high-order byte is sent "On The Wire" before the low-order byte. This is the official 32bit number byte order as specified in the Modbus specification. However most devices appear to ignore this specified byte order using the more conventional "Big" instead

### The Command34 Conversation

Following is the first Conversation in the Modbus protocol definition. This is the Conversation used to implement Modbus Commands "03" and "04". These two Modbus Commands can be used to query most values available on a Modbus Device.

<conversation>

<name value="Command34" />

<message>

<type value="Send" />

<element type="BYTE[1]" name="Address" />

<element type="BYTE[1]" name="CommandId" />

<element type="BYTE[2]" name="FirstAddress" />

<element type="BYTE[2]" name="Registers" />

<element type="BYTE[2]" name="CheckSum16" />

</message>

<message>

<type value="Read" />

<element type="BYTE[1]" name="Address" />

<element type="BYTE[1]" name="CommandId" />

</message>

<message>

<type value="Extract" />

<element type="BYTE[1]" name="DataSize" />

</message>

<message>

<type value="Extract" />

<element type="BYTE[DataSize]" name="Data" />

<element type="BYTE[2]" name="CheckSum16" />

</message>

</conversation>

All elements in this Conversation have sizes that are either fixed or calculated prior to Message execution.

The element named *Address* will be loaded with the 8bit Modbus address of the device as defined in the PVBC device configuration. This element does not need to be referenced from the higher level device definition. It is handled automatically by the PVBC Modbus algorithms.

The element named *CommandId* will be loaded with the 8bit Modbus command value. This is either 0x03 or 0x04. The required value is specified in the higher level device definition.

The element named *FirstAddress* will be loaded with the 16bit Modbus register address of the first Modbus register that will be retrieved by this command. This element does not need to be referenced from the higher level device definition. It is handled automatically by the PVBC Modbus algorithms.

The element named *Registers* will be loaded with the 16bit count of the number of Modbus registers that will be retrieved by this command. This element does not need to be referenced from the higher level device definition. It is handled automatically by the PVBC Modbus algorithms.

The element named *CheckSum16* will be loaded with the 16bit Modbus Checksum. This element does not need to be referenced from the higher level device definition. It is handled automatically by the PVBC Modbus algorithms.

The element named *DataSize* will be loaded with the 8bit data payload size. This element does not need to be referenced from the higher level device definition. It is handled automatically by the PVBC Modbus algorithms.

The element named *Data* will contain the requested Modbus registers after the Conversation has been executed. This element does not need to be referenced from the higher level device definition. It is handled automatically by the PVBC Modbus algorithms. The size of this element is automatically set to the value of the *DataSize* element.

The Conversation uses the first message to send a Modbus command to the device. This command tells the device to respond with the list of register values starting at the specified *FirstAddress* and containing *Registers* register values.

The second message reads the device response up to but excluding the payload size value.

The third message extracts the payload size in bytes from the device response. This value is used to adjust the *Data* element to the size required to contain the data payload.

The fourth message extracts the data payload and response checksum into the *Data* element and *CheckSum16* element. The PVBC Modbus algorithms automatically assign the data payload register values to the appropriate Modbus registers defined in the higher level device definition.

## The SetRegisters Conversation

Following is the second Conversation in the Modbus protocol definition. This is the Conversation used to write values into device registers.

<conversation>

<name value="SetRegisters" />

<message>

<type value ="Send" />

<element type="BYTE[1]" name="Address" />

<element value="0x10" />

<element type="BYTE[2]" name="FirstAddress" />

<element type="BYTE[2]" name="Registers" />

<element type="BYTE[1]" name="DataValueSize" />

<element type="BYTE[DataValueSize]" name="DataValue" />

<element type="BYTE[2]" name="CheckSum16" />

</message>

<message>

<type value="Read" />

<element type="BYTE[1]" name="Address" />

<element value="0x10" />

<element type="BYTE[2]" name="FirstAddress" />

<element type="BYTE[2]" name="Registers" />

<element type="BYTE[2]" name="CheckSum16" />

</message>

</conversation>

All elements in this Conversation have sizes that are either fixed or calculated prior to Conversation execution.

The element named *Address* will be loaded with the 8bit Modbus address of the device as defined in the PVBC device configuration. This element does not need to be referenced from the higher level device definition. It is handled automatically by the PVBC Modbus algorithms.

The 8bit command id element is fixed at a value of 0x10.

The element named *FirstAddress* will be loaded with the 16bit Modbus register address of the first Modbus register that will be set by this command. This element does not need to be referenced from the higher level device definition. It is handled automatically by the PVBC Modbus algorithms.

The element named *Registers* will be loaded with the 16bit count of the number of Modbus registers that will be set by this command. This element does not need to be referenced from the higher level device definition. It is handled automatically by the PVBC Modbus algorithms.

The element named *DataSize* will be loaded with the 8bit data payload size. This element does not need to be referenced from the higher level device definition. It is handled automatically by the PVBC Modbus algorithms.

The element named *DataValue* will be loaded with the values that will set into the specified Modbus registers after the Conversation has been executed. This element does not need to be referenced from the higher level device definition. It is handled automatically by the PVBC Modbus algorithms. The size of this element is automatically set to the value of the *DataSize* element.

The element named *CheckSum16* will be loaded with the 16bit Modbus Checksum. This element does not need to be referenced from the higher level device definition. It is handled automatically by the PVBC Modbus algorithms.

The Conversation uses the first message to send the register settings to the device.

The second message captures the device acknowledgement that the settings were successfully applied.

# Device Definitions

## Query / Response Device Configuration

Following is an example of device configuration using the QueryResponse Protocol. This configuration is the new PVBC Xantrex Non-Modbus device configuration. This device type uses RS232 communication with one serial port required for each inverter monitored by PVBC.

### Device Type Identification

The device definition starts as follows:

<device>

<devicetype value="Inverter" />

<specification value="Xantrex ASCII" />

<version value="01" />

<status value="Test Only" />

<protocol value="QueryResponse\_Generic" />

<devicetype*>* specifies the PVBC Device Model that this device can bind to.

"Inverter" - Inverter is the first and currently the only device type for which there is a device model in PVBC. Others will be added starting with a Meter device type for energy meter configuration

<specification*>* is the name of the device or class of devices supported by this specification. This is used with *<version>* to identify a specific device configuration.

<version*>* is the version number of this device specification . This is used with *<specification>* to identify a specific device configuration.

<status*>* is an optional comment element that identifies the testing status of a specification. This is displayed by the PVBC configuration program as a suffix to the device type identifier. It allows users to identify the configurations that have been verified as working correctly.

<protocol*>* is the name of the protocol specification that is used to implement this device specification. It refers to a protocol defined in the Protocol Specification section of this document.

### Device Interactions

All interactions between PVBC and a device are initiated by the PVBC Device Model selected for the device type. These interactions are classified by an Interaction Type.

The PVBC "Inverter" Device Model understands two Interaction Types:

* Identity - This interaction Type is used at start-up to retrieve the Inverter Make, Model and Serial Number
* Reading - This Interaction Type is used after the identity has been retrieved to take periodic readings from the Inverter

PVBC interacts with devices through "Device Blocks". Each Device Block defines all aspects of a single interaction with a target device. Device Blocks that include an Interaction Type in their specification are activated directly by the PVBC Device Model. This means that for an Inverter, all Device Blocks with an Interaction Type of "Identity" are executed in the order they are defined to retrieve the Inverter identity details.

The Device specification can also contain "Algorithm" definitions. Algorithm definitions can also include an Interaction Type specification. When a PVBC Device Model executes an Interaction Type, all Device Blocks labelled with that interaction type are executed first. Then all Algorithms labelled with that interaction type are executed.

Algorithms encapsulate Device Block execution logic that allow Device Blocks to be executed in specific sequences. Algorithms are only required for devices that require specific sequences of interactions to achieve the required outcomes. Most device types do not require Algorithms, including the simple example examined here.

### The Device Block

The Device Block defines a single interaction with a device. The Device Block shown below retrieves inverter identity information from a Xantrex inverter:

<block>

<name value="Identity" />

<type value="Identity" />

<conversation value="QueryResponseX3" />

<name> provides the Device Block Name. This allows the block to be referenced from Algorithms.

<type> provides the optional Interaction Type of the Device Block. This allows the block to be executed directly by the PCBC Device Model.

<conversation> provides the name of the low level Protocol Conversation that is used to execute this single interaction.

*The Conversation selected above allows a device response to be divided into 3 variable length components based on fixed value "Markers" that delimit the variable content and size response detail.*

### The Register Definition

Within a Device Block, Register Definitions describe the values and variables that are input to or output from the selected Protocol Conversation.

The Register Definition below describes a fixed value that is bound to the Conversation Element named "EndOfMessage". In this Device Definition, "EndOfMessage" is set to the single 8bit byte value 0x0D (Carriage Return). This value is used in all messages to and from the device.

<register>

<type value="byte" />

<message value="literal" />

<registervalue value="0x0D" />

<binding value="EndOfMessage" />

</register>

<type> provides the data type of the data from this register as it is sent "On The Wire". Valid options are:

"bcd" - A Binary Coded Decimal value. The number of digits that can be stored is specified in the <size> element of the Register Definition. The length of the register value "On The Wire" is (number of digits stored + 1) / 2.

"byte" - An 8bit byte value. Length is 1 byte.

"bytes" - An array of 8bit byte values. Length is determined by the value assigned to the register or by a value in the <size> element of the Register Definition.

"cstring" - An array of ASCII characters stored as 8bit byte values. Length is determined by the value assigned to the register or by a value in the <size> element of the Register Definition. If the assigned value is shorter than the specified size, the value will be right padded with binary zero (0x00).

"sint16" - A 16bit signed binary value. This has a length of 2 bytes.

"sint32" - A 32bit signed binary value. This has a length of 4 bytes.

"uint16" - A 16bit unsigned binary value. This has a length of 2 bytes.

"uint32" - A 32bit unsigned binary value. This has a length of 4 bytes.

"string" - An array of ASCII characters stored as 8bit byte values. Length is determined by the value assigned to the register or by a value in the <size> element of the Register Definition. If the assigned value is shorter than the specified size, the value will be right padded with spaces.

<message> provides the Protocol Conversation Message Type that this register is associated with. Valid options are:

"send" - This register provides values used to send messages to the device.

"receive" - This register receives values extracted from a message received from the device.

"literal" - This register provides a literal (fixed) value that can be used in all messages.

"both" - This register provides values used to send messages to the device. It also receives values extracted from a message received from the device.

<binding> provides the name of the Protocol Conversation Element that this register is bound to. Data is transferred between the Register and the Conversation Element as required.

The Register Definition below describes a string value that is bound to the Conversation Element named "QueryString". In this Device Block definition, "QueryString" is set to "IDN?". This is the Xantrex command that requests the inverter identification details.

<register>

<type value="string" />

<message value="send" />

<registervalue value="IDN?" />

<binding value="QueryString" />

</register>

<registervalue> provides the string representation of the value to be assigned to the Register.

These first two registers provide all information required to allow the Protocol Conversation to send this request to the device. The remaining Registers in this Device Block are used to identify the three variable length components of the device response.

<register>

This register provides the literal value ("M:") that precedes the Model in the device response .

It is bound to the Conversation Element named "Marker1".

<type value="string" />

<message value="literal" />

<registervalue value="M:" />

<binding value="Marker1" />

</register>

<register>

This register provides the link between the Conversation Element that will be loaded with Model from the device response, and the PVBC Device Model variable that will process the value as required by the PVBC Device Model algorithms.

<type value="string" />

<message value="receive" />

<content value="Model" />

<binding value="Response1" />

</register>

<content> provides the name of the PVBC Device Model (Inverter) variable that this register is bound to. Data is transferred between the Register and the Device Model variable as required.

<register>

This register provides the literal value (" X:") that separates Model from the Xanbus Id in the device response .

It is bound to the Conversation Element named "Marker2".

<type value="string" />

<message value="literal" />

<registervalue value=" X:" />

<binding value="Marker2" />

</register>

<register>

This register provides the link between the Conversation Element that will be loaded with Xanbus Id from the device response. This value is not required by the PVBC Device Model. Consequently there is no binding to a Model variable.

<type value="string" />

<message value="receive" />

<binding value="Response2" />

</register>

<register>

This register provides the literal value (" S:") that separates the Xanbus Id from the Serial Number in the device response .

It is bound to the Conversation Element named "Marker3".

<type value="string" />

<message value="literal" />

<registervalue value=" S:" />

<binding value="Marker3" />

</register>

<register>

This register provides the link between the Conversation Element that will be loaded with Serial Number from the device response, and the PVBC Device Model variable that will process the value as required by the PVBC Device Model algorithms.

<type value="string" />

<message value="receive" />

<content value="SerialNo" />

<binding value="Response3" />

</register>

<register>

This register provides the literal value ("") that follows Serial Number in the device response .

It is bound to the Conversation Element named "Marker4".

<type value="string" />

<message value="literal" />

<registervalue value="" />

<binding value="Marker4" />

</register>

</block>

These last five registers provide all information required to allow the Protocol Conversation to receive a response from the device and deliver the two required variable values to the PVBC Device Model for processing.

Following is a Device Block that will retrieve an AC Power reading from a Xantrex inverter:

<block>

<name value="Power" />

<type value="Reading" />

<conversation value="QueryResponse" />

<register>

<type value="string" />

<message value="send" />

<registervalue value="POUT?" />

<binding value="QueryString" />

</register>

<register>

<type value="string" />

<message value="receive" />

<content value="PowerAC1" />

<binding value="Response" />

</register>

</block>

Following is a Device Block that will retrieve a Temperature reading from a Xantrex inverter. The request retrieves the temperature in both Centigrade and Fahrenheit. Only the Centigrade value is bound to the PVBC Device Model.

<block>

<name value="Temperature" />

<type value="Reading" />

<conversation value="QueryResponseX2" />

<register>

<type value="string" />

<message value="send" />

<registervalue value="MEASTEMP?" />

<binding value="QueryString" />

</register>

<register>

<type value="string" />

<message value="literal" />

<registervalue value="C:" />

<binding value="Marker1" />

</register>

<register>

<type value="string" />

<message value="receive" />

<content value="Temperature" />

<binding value="Response1" />

</register>

<register>

<type value="string" />

<message value="literal" />

<registervalue value=" F:" />

<binding value="Marker2" />

</register>

<register>

<type value="string" />

<message value="receive" />

<binding value="Response2" />

</register>

<register>

<type value="string" />

<message value="literal" />

<registervalue value="" />

<binding value="Marker3" />

</register>

</block>

<devices>

<device>

<devicetype value="Inverter" />

<specification value="Xantrex" />

<version value="01" />

<status value="Test Only" />

<protocol value="Modbus" />

<endian16bit value="Big" />

<endian32bit value="Big" />

<typeoptions>

<type value="uint16" />

<valuelist>

<option value="65535" tag="NotAvailable" />

<option value="65534" tag="Invalid" />

</valuelist>

</typeoptions>

<typeoptions>

<type value="sint16" />

<valuelist>

<option value="32767" tag="NotAvailable" />

<option value="32766" tag="Invalid" />

</valuelist>

</typeoptions>

<block>

<name value="Identity" />

<type value="Identity" />

<conversation value="Command34" />

<commandid value="3" />

<base value="0" />

<register>

<content value="Manufacturer" />

<type value="string" />

<registervalue value="Xantrex" />

</register>

<register>

<content value="Model" />

<id value="0x0000" />

<type value="cstring" />

<size value="20" />

</register>

<register>

<content value="SerialNo" />

<id value="0x0014" />

<type value="cstring" />

<size value="20" />

</register>

</block>

<block>

<name value="Inverter Status" />

<type value="Reading" />

<ondbwriteonly value="true" />

<conversation value="Command34" />

<commandid value="3" />

<base value="0" />

<register>

<content value="Status" />

<id value="0x00CF" />

<type value="uint16" />

</register>

</block>

<block>

<name value="Inverter DC" />

<type value="Reading" />

<ondbwriteonly value="true" />

<conversation value="Command34" />

<commandid value="3" />

<base value="0" />

<register>

<content value="VoltsPV1" />

<id value="0x0201" />

<type value="uint32" />

<scale value="100" />

</register>

<register>

<content value="CurrentPV1" />

<id value="0x0203" />

<type value="uint32" />

<scale value="100" />

</register>

</block>

<block>

<name value="Inverter AC" />

<type value="Reading" />

<conversation value="Command34" />

<commandid value="3" />

<base value="0" />

<register>

<content value="VoltsAC1" />

<id value="0x0701" />

<type value="uint32" />

<scale value="100" />

</register>

<register>

<content value="CurrentAC1" />

<id value="0x0703" />

<type value="uint32" />

<scale value="100" />

</register>

<register>

<content value="PowerAC1" />

<id value="0x0706" />

<type value="uint32" />

<scale value="1" />

</register>

<register>

<content value="Frequency" />

<id value="0x0705" />

<type value="uint16" />

<scale value="10" />

</register>

</block>

<block>

<name value="Temperature" />

<type value="Reading" />

<ondbwriteonly value="true" />

<conversation value="Command34" />

<commandid value="3" />

<base value="0" />

<register>

<content value="Temperature" />

<id value="0x0900" />

<type value="sint16" />

<scale value="10" />

</register>

</block>

<block>

<name value="Error Count" />

<type value="Reading" />

<ondbwriteonly value="true" />

<conversation value="Command34" />

<commandid value="3" />

<base value="0" />

<register>

<content value="ErrorCode" />

<iserrorflag value="true" />

<erroraction>

<type value="CallAlgorithm" />

<algorithm value="Log Errors" />

</erroraction>

<id value="0x0080" />

<type value="uint16" />

<valuelist>

<option value="0" tag="OK" />

</valuelist>

</register>

</block>

<block>

<name value="Initialise Error Retrieval" />

<conversation value="SetRegisters" />

<base value="0" />

<register>

<name value="ErrorId" />

<id value="0x0081" />

<type value="uint16" />

<registervalue value="0" />

</register>

</block>

<block>

<name value="Retrieve Error" />

<conversation value="Command34" />

<commandid value="3" />

<base value="0" />

<register>

<name value="ErrorValue" />

<id value="0x0082" />

<type value="bytes" />

<size value="8" />

</register>

<register>

<name value="ErrorDesc" />

<id value="0x0086" />

<type value="cstring" />

<size value="40" />

</register>

</block>

<block>

<name value="Initialise History Retrieval" />

<conversation value="SetRegisters" />

<base value="0" />

<register>

<name value="HistoryType" />

<id value="0x0800" />

<type value="uint16" />

</register>

<register>

<name value="Options" />

<id value="0x0801" />

<type value="bytes" />

<size value="4" />

<registervalue value="0x00 00 00 00" />

</register>

<register>

<name value="HistoryId" />

<id value="0x0803" />

<type value="uint16" />

</register>

</block>

<block>

<name value="Retrieve History" />

<conversation value="Command34" />

<commandid value="3" />

<base value="0" />

<register>

<name value="HistoryId" />

<id value="0x0803" />

<type value="uint16" />

</register>

<register>

<name value="Energy" />

<id value="0x0804" />

<type value="sint32" />

<scale value="10" />

</register>

<register>

<name value="PeakPower" />

<id value="0x0806" />

<type value="sint32" />

<scale value="1" />

</register>

<register>

<name value="HarvestTime" />

<id value="0x0808" />

<type value="uint32" />

<scale value="1" />

</register>

</block>

<algorithm>

<type value="Reading" />

<ondbwriteonly value="true" />

<name value="Retrieve EnergyTodayAC" />

<action>

<type value="SendBlock" />

<blockname value="Initialise History Retrieval" />

<parameter>

<name value="HistoryType" />

<parametervalue value="4" />

</parameter>

<parameter>

<name value="HistoryId" />

<parametervalue value="0" />

</parameter>

</action>

<action>

<type value="GetBlock" />

<blockname value="Retrieve History" />

<parameter>

<name value="Energy" />

<content value="EnergyTodayAC" />

</parameter>

</action>

</algorithm>

<algorithm>

<type value="Reading" />

<ondbwriteonly value="true" />

<name value="Retrieve EnergyTotalAC" />

<action>

<type value="SendBlock" />

<blockname value="Initialise History Retrieval" />

<parameter>

<name value="HistoryType" />

<parametervalue value="5" />

</parameter>

<parameter>

<name value="HistoryId" />

<parametervalue value="0" />

</parameter>

</action>

<action>

<type value="GetBlock" />

<blockname value="Retrieve History" />

<parameter>

<name value="Energy" />

<content value="EnergyTotalAC" />

</parameter>

<parameter>

<name value="HarvestTime" />

<content value="TimeTotal" />

</parameter>

</action>

</algorithm>

<algorithm>

<type value="ErrorLog" />

<name value="Log Errors" />

<action>

<type value="SendBlock" />

<blockname value="Initialise Error Retrieval" />

</action>

<action>

<type value="RepeatCountTimes" />

<blockname value="Retrieve Error" />

<count value="ErrorCode" />

<action>

<type value="GetBlock" />

</action>

<action>

<type value="LogError" />

</action>

</action>

</algorithm>

</device>

<device>

<devicetype value="Inverter" />

<specification value="Xantrex ASCII" />

<version value="01" />

<status value="Test Only" />

<protocol value="QueryResponse\_Generic" />

<block>

<name value="Identity" />

<type value="Identity" />

<conversation value="QueryResponseX3" />

<register>

<type value="byte" />

<message value="literal" />

<registervalue value="0x0D" />

<binding value="EndOfMessage" />

</register>

<register>

<type value="string" />

<message value="send" />

<registervalue value="IDN?" />

<binding value="QueryString" />

</register>

<register>

<type value="string" />

<message value="literal" />

<registervalue value="M:" />

<binding value="Marker1" />

</register>

<register>

<type value="string" />

<message value="receive" />

<content value="Model" />

<binding value="Response1" />

</register>

<register>

<type value="string" />

<message value="literal" />

<registervalue value=" X:" />

<binding value="Marker2" />

</register>

<register>

<type value="string" />

<message value="receive" />

<binding value="Response2" />

</register>

<register>

<type value="string" />

<message value="literal" />

<registervalue value=" S:" />

<binding value="Marker3" />

</register>

<register>

<type value="string" />

<message value="receive" />

<content value="SerialNo" />

<binding value="Response3" />

</register>

<register>

<type value="string" />

<message value="literal" />

<registervalue value="" />

<binding value="Marker4" />

</register>

</block>

<block>

<name value="Power" />

<type value="Reading" />

<conversation value="QueryResponse" />

<register>

<type value="string" />

<message value="send" />

<registervalue value="POUT?" />

<binding value="QueryString" />

</register>

<register>

<type value="string" />

<message value="receive" />

<content value="PowerAC1" />

<binding value="Response" />

</register>

</block>

<block>

<name value="Temperature" />

<type value="Reading" />

<conversation value="QueryResponseX2" />

<register>

<type value="string" />

<message value="send" />

<registervalue value="MEASTEMP?" />

<binding value="QueryString" />

</register>

<register>

<type value="string" />

<message value="literal" />

<registervalue value="C:" />

<binding value="Marker1" />

</register>

<register>

<type value="string" />

<message value="receive" />

<content value="Temperature" />

<binding value="Response1" />

</register>

<register>

<type value="string" />

<message value="literal" />

<registervalue value=" F:" />

<binding value="Marker2" />

</register>

<register>

<type value="string" />

<message value="receive" />

<content value="TemperatureF" />

<binding value="Response2" />

</register>

<register>

<type value="string" />

<message value="literal" />

<registervalue value="" />

<binding value="Marker3" />

</register>

</block>

<block>

<name value="Inverter Inputs" />

<type value="Reading" />

<conversation value="QueryResponseX3" />

<register>

<type value="string" />

<message value="send" />

<registervalue value="MEASIN?" />

<binding value="QueryString" />

</register>

<register>

<type value="string" />

<message value="literal" />

<registervalue value="V:" />

<binding value="Marker1" />

</register>

<register>

<type value="string" />

<message value="receive" />

<content value="VoltsPV1" />

<binding value="Response1" />

</register>

<register>

<type value="string" />

<message value="literal" />

<registervalue value=" I:" />

<binding value="Marker2" />

</register>

<register>

<type value="string" />

<message value="receive" />

<content value="CurrentPV1" />

<binding value="Response2" />

</register>

<register>

<type value="string" />

<message value="literal" />

<registervalue value=" P:" />

<binding value="Marker3" />

</register>

<register>

<type value="string" />

<message value="receive" />

<content value="PowerPV1" />

<binding value="Response3" />

</register>

<register>

<type value="string" />

<message value="literal" />

<registervalue value="" />

<binding value="Marker4" />

</register>

</block>

<block>

<name value="Inverter Outputs" />

<type value="Reading" />

<conversation value="QueryResponseX4" />

<register>

<type value="string" />

<message value="send" />

<registervalue value="MEASOUT?" />

<binding value="QueryString" />

</register>

<register>

<type value="string" />

<message value="literal" />

<registervalue value="V:" />

<binding value="Marker1" />

</register>

<register>

<type value="string" />

<message value="receive" />

<content value="VoltsAC1" />

<binding value="Response1" />

</register>

<register>

<type value="string" />

<message value="literal" />

<registervalue value=" I:" />

<binding value="Marker2" />

</register>

<register>

<type value="string" />

<message value="receive" />

<content value="CurrentAC1" />

<binding value="Response2" />

</register>

<register>

<type value="string" />

<message value="literal" />

<registervalue value=" P:" />

<binding value="Marker3" />

</register>

<register>

<type value="string" />

<message value="receive" />

<content value="PowerAC1" />

<binding value="Response3" />

</register>

<register>

<type value="string" />

<message value="literal" />

<registervalue value=" F:" />

<binding value="Marker4" />

</register>

<register>

<type value="string" />

<message value="receive" />

<content value="Frequency" />

<binding value="Response4" />

</register>

<register>

<type value="string" />

<message value="literal" />

<registervalue value="" />

<binding value="Marker5" />

</register>

</block>

<block>

<name value="EnergyToday" />

<type value="Reading" />

<ondbwriteonly value="true" />

<conversation value="QueryResponse" />

<register>

<type value="string" />

<message value="send" />

<registervalue value="KWHTODAY?" />

<binding value="QueryString" />

</register>

<register>

<type value="string" />

<message value="receive" />

<content value="EnergyTodayAC" />

<binding value="Response" />

</register>

</block>

<block>

<name value="EnergyTotal" />

<type value="Reading" />

<ondbwriteonly value="true" />

<conversation value="QueryResponse" />

<register>

<type value="string" />

<message value="send" />

<registervalue value="KWHLIFE?" />

<binding value="QueryString" />

</register>

<register>

<type value="string" />

<message value="receive" />

<content value="EnergyTotalAC" />

<binding value="Response" />

</register>

</block>

</device>

</devices>

# Protocol Reference

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Element | | Parent | Attributes | Values | Description |
| <devicemanagement> .... </devicemanagement> | | |  |  | Contains all protocol definitions. It can also contain some or all device definitions |
| <protocol> .... </protocol> | | devicemanagement |  |  | Contains all elements required to describe one complete Protocol |
|  | **<name ... />** | protocol | value |  | Provides the name of a Protocol |
|  | **<type .../>** | protocol |  |  | Provides the name of a Protocol Type |
|  |  |  | value | *QueryResponse* | Selects the PVBC general purpose Query / Response protocol algorithm |
|  |  |  |  | *Modbus* | Selects the PVBC Modbus specific protocol algorithm |
| <conversation> ... <conversation> | | protocol |  |  | Contains all elements required to describe one complete Conversation |
|  | **<name ... />** | conversation | value |  | Provides the name of a Conversation |
| <message>... </message> | | conversation |  |  | Contains all elements required to describe one Message in a Conversation |
|  | **<type .../>** | message |  |  | Provides the name of a Message Type |
|  |  |  | value | *Send* | This message type is constructed and sent to the device. The message succeeds if it is sent successfully |
|  |  |  |  | *Read* | This message type provides a fixed pattern that is compared with bytes read from the input stream. The message succeeds if the bytes in the input stream match the content of this message. It fails if an input stream timeout occurs or if the bytes in the input stream do not match the pattern described by the elements in this message |
|  |  |  |  | *Extract* | This message type copies bytes into the elements of the message from the input stream. All elements in the message must be variables unless the first element is of type DYNAMICBYTE. Refer to Element type DYNAMICBYTE for further details. The message succeeds when all elements in the message have been assigned values. The message fails if a timeout occurs on the input stream |
|  |  |  |  | *Find* | This message type skips all bytes in the input stream until the pattern defined by the elements in the message are found. The message succeeds when a match is found. It fails if the input stream read timeout is reached or if the skip limit is reached |
|  | **<element .../>** | message |  |  |  |
|  |  |  | name |  | Provides a name for this element. Named elements can be referenced from the higher level Device definitions |
|  |  |  | type | *BYTE* | An array of 8bit bytes can be stored in this element. Type BYTE with no length specified has its length determined by the value assigned to it. A value must be assigned before this element type can be used as it depends on the assignment to determine the size.  This is the default type if no type is specified |
|  |  |  |  | *BYTE[n]* | An array of 8bit bytes can be stored in this element. Type BYTE with a fixed size specified can be used without prior assignment in an Extract message. It requires prior assignment for use in other message types or when used in after a DYNAMICBYTE element in an Extract message type |
|  |  |  |  | *BYTE[name]* | An array of 8bit bytes can be stored in this element. Type BYTE with the name of another element as the length specifier will have its length automatically adjusted whenever the value in the named element changes. The named element must be of type BYTE with a length of 1, 2 or 4 bytes. The named element will be interpreted as an unsigned binary integer in all cases |
|  |  |  |  | *STRING* | An array of ASCII characters (8bit) can be stored in this element. A value must be assigned before this element type can be used as it depends on the assignment to determine the size |
|  |  |  |  | *DYNAMICBYTE* | An array of 8bit bytes can be stored in this element. This type is only used in messages of type "Extract". It must be the first element in the message. All other elements in the message must have assigned values. The DYNAMICBYTE element is filled with all bytes read from the input stream until bytes that match the pattern defined by the other elements in the message are found. The final size of the element is the number of bytes read from the input stream before a match was found |
|  |  |  | value | *"0x...." or "0X...."* | Assigns a value to the element. If the element is of type "BYTE", the specified string value is treated as a hexadecimal representation of the byte array to be assigned. If the element is of type "STRING", the assignment is literal without hexadecimal interpretation.  If the element type has a size specified, that size is ignored. The size is determined by the size of the value assigned |
|  |  |  |  | *"...."* | Assigns a value to the element as an array of ASCII characters represented by the string value specified. This applies to both BYTE and STRING element types.  If the element type has a size specified, that size is ignored. The size is determined by the size of the value assigned |

# Device Definition Reference

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Element | | Parent | Attributes | Values | Description |
| <devices> .... </devices> | | |  |  | Container for device definitions. |
| <device> .... </device> | | devices |  |  | Contains all elements required to describe one complete Device |
|  | **<devicetype ... />** | device |  |  | Identifies the PVBC Device Model that can drive this device configuration |
|  |  |  | value | *Inverter* | Extract and process data that meets the requirements of a PVBC Inverter. Note that Inverter is the only supported type at this time |
|  | **<specification .../>** | device | value |  | The name of the device or class of devices supported by this specification. This is used with *<version>* to identify a specific device configuration |
|  | **<version ... />** | device | value |  | The version number of this device specification . This is used with *<specification>* to identify a specific device configuration |
|  | **<status ... />** | device | value |  | An optional comment element that identifies the testing status of a specification. This is displayed by the PVBC configuration program as a suffix to the device type identifier. It allows users to identify the configurations that have been verified as working correctly |
|  | **<protocol ... />** | device | value |  | The name of the protocol specification that is used to implement this device specification. |
| <block>... </block> | | device |  |  | Contains all elements required to describe one Device Block in a Device |
|  | **<base .../>** | block | value |  | The Modbus Register base address used by this block. The "On The Wire" Modbus register address is calculated by subtracting the <base> value from the Device Block Register <id> value. This value can be provided as either a decimal value or a hexadecimal value.  Modbus Device specifications are generally written using Register Addresses that imply the Modbus Command Id that should be used to access them. These specified Register Addresses are generally different to the the |
|  | **<name .../>** | block | value |  | The Device Block Name. This allows the block to be referenced from Algorithms |
|  | **<ondbwriteonly .../>** | block | value |  | This block is only executed directly by the PVBC Device Model on device readings at which data is due to be recorded in the database.  This can be used to reduce device activity when a short query cycle is used to obtain frequent power readings. Energy, voltage and current readings are only required when data is being recorded in the PVBC database |
|  | **<type .../>** | block | value |  | The optional Interaction Type of the Device Block. This allows the block to be executed directly by the PCBC Device Model |
|  | **<conversation .../>** | block | value |  | the name of the low level Protocol Conversation that is used to execute this single interaction (Device Block) |
| <register>... </register> | | device |  |  | Contains all elements required to describe one Register in a Device Block |
|  | **<binding .../>** | register |  |  | The name of the Protocol Conversation Element that this register is bound to. Data is transferred between the Register and the Conversation Element as required |
|  | **<content .../>** | register | value |  | The name of the PVBC Device Model (Inverter) variable that this register is bound to. Data is transferred between the Register and the Device Model variable as required |
|  | **<id .../>** | register | value |  | The Modbus Register ID accessed by this register. This is only used with Modbus devices using the PVBC Modbus protocol. The value can be provided as either a decimal value or a hexadecimal value ("0x...") |
|  | **<message .../>** | register |  |  | The Protocol Conversation Message Type that this register is associated with |
|  |  |  | value | *send* | This register provides values used to send messages to the device |
|  |  |  | value | *receive* | This register receives values extracted from a message received from the device |
|  |  |  | value | *literal* | This register provides a literal (fixed) value that can be used in all messages |
|  |  |  | value | *both* | This register provides values used to send messages to the device. It also receives values extracted from a message received from the device |
|  | **<registervalue .../>** | register | value |  | The string representation of the value to be assigned to the Register |
|  | **<scale .../>** | register | value |  | Used when the register is assigned a numeric value that includes a number of implied decimal places. That is a fractional value is to be represented by an integer with an inplied specific number of decimal places. The scale vale is 10 ^ n where n is the number of implied decimal places.  10 - One decimal place; 100 - Two decimal places |
|  | **<size .../>** | register | value |  | The size of the data value that can be stored in this register. This is used with Registers of type "bytes", "cstring" and "string" when a fixed length value is required |
|  | **<type .../>** | register |  |  | The data type of the data from this register as it is sent "On The Wire" |
|  |  |  | value | *bcd* | A Binary Coded Decimal value. The number of digits that can be stored is specified in the <size> element of the Register Definition. The length of the register value "On The Wire" is (number of digits stored + 1) / 2 |
|  |  |  | value | *byte* | An 8bit byte value. Length is 1 byte |
|  |  |  | value | *bytes* | An array of 8bit byte values. Length is determined by the value assigned to the register or by a value in the <size> element of the Register Definition. If an assigned value is shorter than the specified size, the value will be right padded with zeros 0x0 |
|  |  |  | value | *cstring* | An array of ASCII characters stored as 8bit byte values. Length is determined by the value assigned to the register or by a value in the <size> element of the Register Definition. If the assigned value is shorter than the specified size, the value will be right padded with binary zero (0x00) |
|  |  |  | value | *sint16* | A 16bit signed binary value. This has a length of 2 bytes |
|  |  |  | value | *sint32* | A 32bit signed binary value. This has a length of 4 bytes |
|  |  |  | value | *uint16* | A 16bit unsigned binary value. This has a length of 2 bytes |
|  |  |  | value | *uint32* | A 32bit unsigned binary value. This has a length of 4 bytes |
|  |  |  | value | *string* | An array of ASCII characters stored as 8bit byte values. Length is determined by the value assigned to the register or by a value in the <size> element of the Register Definition. If the assigned value is shorter than the specified size, the value will be right padded with spaces |