-

A Sierra Monitor Company

Driver Manual
(Supplement to the FieldServer Instruction Manual)

FS-8700-116 Wattmaster for ProtoCessor - Serial Driver

APPLICABILITY & EFFECTIVITY

Effective for all systems manufactured after May 1, 2001

Driver Version:	**0.02**
Document Revision:	**1**

TABLE OF CONTENTS

1. Wattmaster Serial Driver Description

The Wattmaster Serial Protocol is designed to achieve the following

- Read Customer Device Database of data objects
- Determine if the database has changed
- Allow data transfer (read and write) between the driver and the customer Device

This driver utilizes these protocol capabilities to achieve the following

- Read customer database
- Autocreate a Client configuration to read the values/states of the data objects discovered
- Autocreate a Server side connection, node and objects to make the customer device available to upstream Clients.
- Various types of Server are available
    - BACnet IP / Ethernet / MSTP
    - Metasys N2 Open
    - LonWorks

The driver can also be used in a simple mode to poll for specific data from a customer device. To achieve this, the data objects in the customer device must be known in advance.

The protocol is node-less. Messages do not contain source/destination node addresses and it is not possible to differentiate messages from multiple devices on the same connection. Thus only one node may be defined per connection.
The driver provides both Client and Server emulation. The Server side of the driver is intended to support FieldServer's Quality Assurance program and is not intended to provide complete emulation of a Wattmaster Server. Thus the Server side is not fully documented. However, at a customer's request the Server side functionality can be documented and enhanced - please contact FieldServer's sales group.

The protocol specification document is available in Appendix C

Max Nodes Supported

FieldServer Mode	Nodes	Comments
Client	1	1 Node per serial port
Server	1	1 Node per serial port

2. Driver Scope of Supply

2.1. Supplied by FieldServer Technologies for this driver

FieldServer Technologies PART #	Description
FS-8915-10	UTP cable (7 foot) for Ethernet connection
FS-8915-10	UTP cable (7 foot) for RS-232 use
FS-8917-02	RJ45 to DB9F connector adapter
FS-8700-109	Driver Manual.

2.2. Provided by the Supplier of 3rd Party Equipment

2.2.1. Required 3rd Party Hardware

This driver is intended for use with a ProtoCessor which has been previously integrated into a customer's controller. No additional hardware is required.

2.2.2. Required 3rd Party Software

This driver is intended for use with a ProtoCessor which has been previously integrated into a customer's controller. The customer's controller must have been previously programmed to implement the Wattmaster protocol. No additional software is required.

2.2.3. Required 3rd Party Configuration

This driver is intended for use with a ProtoCessor which has been previously integrated into a customer's controller. The customer's controller must have been previously programmed to implement the Wattmaster protocol. Discussion of the customer's hardware/software configuration is beyond the scope of this document.

2.2.4. Optional Items

None.

3. Hardware Connections

The ProtoCessor is integrated into the customer's hardware at design time. No additional connection information is required.

3.1. Hardware Connection Tips/Hints

None.

4. Configuring the FieldServer as a Wattmaster Client

For a detailed discussion on FieldServer configuration, please refer to the FieldServer Configuration Manual. The information that follows describes how to expand upon the factory defaults provided in the configuration files included with the FieldServer (See ".csv" sample files provided with the FieldServer).

This section documents and describes the parameters necessary for configuring the FieldServer to communicate with a Wattmaster protocol enabled device.

4.1. Data Arrays/Descriptors

The configuration file tells the FieldServer about its interfaces, and the routing of data required. In order to enable the FieldServer for Wattmaster Serial Driver communications, the driver independent FieldServer buffers need to be declared in the "Data Arrays" section, the destination device addresses need to be declared in the "Client Side Nodes" section, and the data required from the Servers needs to be mapped in the "Client Side Map Descriptors" section. Details on how to do this can be found below.

Note that in the tables, * indicates an optional parameter, with the **bold** legal value being the default.

Section Title
Data_Arrays

Column Title	Function	Legal Values
Data_Array_Name	Provide name for Data Array	Up to 15 alphanumeric characters
Data_Array_Format	Provide data format. Each Data Array can only take on one format.	Float, Bit, UInt16, SInt16, Packed_Bit, Byte, Packed_Byte, Swapped_Byte
Data_Array_Length	Number of Data Objects. Must be larger than the data storage area required by the Map Descriptors for the data being placed in this array.	1-10,000

Example

```
//    Data Arrays

Data_Arrays
Data_Array_Name,            Data_Format,          Data_Array_Length
DA_AI_01,                   UInt16,               200
DA_AO_01,                   UInt16,               200
DA_DI_01,                   Bit,                  200
DA_DO_01,                   Bit,                  200
```

4.2. Client Side Connection Descriptions

Section Title
Connections

Column Title	Function	Legal Values
Port	Specify which port the device is connected to the FieldServer	P1-P8, R1-R2[1]
Protocol	Specify protocol used	Wattmaster , wattmstr
Baud*	Specify baud rate Default Baud rate on a ProtoCessor is 38400. Default on a FieldServer is 9600	Driver Supports: 110; to 115200 Standard baud rates only. **38400**
Parity*	Specify parity	Driver Supports: Odd, **Even**, None
Data_Bits*	Specify data bits	Driver Supports: 7,**8**
Stop_Bits*	Specify stop bits	Driver Supports: **1**,2
Handshaking*	Specify hardware handshaking	RTS, RTS/CTS, **None**
Poll _Delay*	Time between internal polls	0-32000 s, **1 s**
Auto_Config_Client*	Omit or set to 'No' if not required. If set to Yes, the driver will auto create Data Arrays and Map Descriptors to read the values/states of the objects discovered in the downstream node.	Yes, Fast, No Additional notes are provided in Appendix A.1
Auto_Config_Server*	Omit or set to 'No' if not required. If specified, the driver auto creates Server Side Connections, Nodes and Map Descriptors to serve the data read by the auto configured Client. The Client side needs to be auto created to use this option.	BACnet-IP, BACnet-Ethernet, BACnet-MSTP, MN2, Lonworks, No Refer to Appendix A.2
Extra_Timeout_Control*	To reduce the number of errors displayed set this parameter to 1. This causes the driver to re-poll once per timeout before the timeout becomes an official error.	No, 1 See Appendix A.5
Server_Object_ID_Style*	Omitting this parameter or specifying a value of zero (default) gives the Server side BACnet objects sequential Object ID's. When the parameter is set to 1 then the driver uses the class, instance and property number to number the Server side objects.	**0**,1,2,3 Additional notes are provided in Appendix A.3

[1] Not all ports shown are necessarily supported by the hardware. Consult the appropriate Instruction manual for details of the ports available on specific hardware.

Example 1

Basic example with no advanced features: Suitable for polling a downstream node where the database is known in advance.

```
//        Client Side Connections

Connections
Port,          Protocol,          Baud,     Parity
P8,            Wattmaster ,       38400,    None
```

Example 2

Advanced example: With a suitable Map Descriptor to poll the database, the driver will read the database, on completion it will create Data Arrays and Map Descriptors to poll for values/states for each object/property discovered and it will also create Server Side connection, node and Map descriptors to serve this data to remote BACnet Clients.

```
//        Client Side Connections

Connections
Port,    Protocol,        Baud,     Parity,    Auto_Config_Client,     Auto_Config_Server
P8,      Wattmaster ,     38400,    None,      Yes,                    BACnet-IP
```

4.3. Client Side Node Descriptors

Section Title
Nodes

Column Title	Function	Legal Values
Node_Name	Provide name for node	Up to 32 alphanumeric characters
Node_ID	This parameter is used when a Server-Side is auto-created.	Refer to Appendix A.2
Protocol	Specify protocol used	Wattmaster , wattmstr
Connection	Specify which port the device is connected to the FieldServer	P1-P8, R1-R2[2]

Example

```
//    Client Side Nodes

Nodes
Node_Name,              Protocol,             Connection
Controller1             Wattmaster,           P8
```

[2] Not all ports shown are necessarily supported by the hardware. Consult the appropriate Instruction manual for details of the ports available on specific hardware.

4.4. Client Side Map Descriptors

4.4.1. FieldServer Related Map Descriptor Parameters

Column Title	Function	Legal Values
Map_Descriptor_Name	Name of this Map Descriptor	Up to 32 alphanumeric characters
Data_Array_Name	Name of Data Array where data is to be stored in the FieldServer	One of the Data Array names from "Data Array" section above
Data_Array_Offset	Starting location in Data Array	0 to maximum specified in "Data Array" section above
Function	Function of Client Map Descriptor.	Use read commands like rdbc, rdb, ars for commands 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x11, 0x12, Use write commands like wrbc, wrbx for commands 0x21

4.4.2. Driver Related Map Descriptor Parameters

Column Title	Function	Legal Values
Node_Name	Name of Node to fetch data from	One of the node names specified in "Client Node Descriptor" above
Data_Type	This commonly used parameter is ignored by this driver.	
Length	Length of Map Descriptor. Each Map Descriptor defines storage locations for a series of addresses. This parameter specifies the length of the series.	1,2,3 .etc Whole numbers
Address	This commonly used parameter is ignored by this driver.	
Wattmstr_Data_Type	When data is transferred between the Wattmaster Server and Client, the value or status for each property is transferred using a 2 byte field. This parameter tells the driver how to interpret this field. Used to define the Server object type when the driver auto creates a Server. Tells the driver how to unpack the field when responding to polls. Tells the driver how to prepare the two byte field when sending data. Refer to Appendix A.4	BIT, BYTE, UINT, SINT, F.1, F.2 , F.3

Column Title	Function	Legal Values
Cmd	Specifies the command that must be executed by the Map Descriptor. The protocol specification document provides additional details. A Command Summary is provided in Appendix A.6 A special command is used by the driver to read the database (0x00). This command is not defined in the protocol spec but rather instructs the driver to proceed through a sequence of commands 0x01...0x06 to read the full database from the Wattmaster Server.	Can be specified in Hex e.g. 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x11, 0x12, 0x21 Can be Specified in Decimal 1, 2, 3, 4, 5, 6, 17, 18, 33
Class_Type	The meaning of this parameter is context sensitive. For commands 0x01 to 0x06 the parameter defines an Index number e.g. The 0'th Class, the 1st Class, etc For commands 0x11,0x12, 0x21 the parameter defines the Class Number e.g. Class whose number is 1, class whose number is 2 …	Whole Numbers
Inst_Num	The class instance number. This parameter is only required by some commands	Whole Numbers
Prop_Num	The property number being read/written. Only required for commands 0x11, 0x12, 0x21	Whole Number
Prop_Index	Used by command 0x05. An index number e.g. the 0th property, the 1st property	Whole Numbers
AutoCreated	When the driver auto-creates Client side Map Descriptors it marks them. This is intended as a trouble shooting tool and should not be included in the configuration file.	Yes, No

4.4.3. Timing Parameters

Column Title	Function	Legal Values
Scan_Interval	Rate at which data is polled	≥0.001s

4.4.4. Map Descriptor Example 1 – Read Database

The Cmd = 0x00 tells the driver to read the Wattmaster Server's database. The driver will use commands 0x01.0x06 to achieve this. The driver will dump the database in ASCII in the specified Data Array. View the array in 'String format.

```
//    Client Side Map Descriptors

Map_Descriptors
Map_Descriptor_Name,      Data_Array_Name,      Data_Array_Offset,      Function,    Node_Name,    Length,    Cmd
Read_Database             Wattmstr-dump,        0,                      Rdbc,        CPU1,         2000,      0x00
```

If the database requires more space it will be truncated. Set the length to 1 if you are not interested in the using the dump.

This command tells the driver to read the database.

4.4.5. Map Descriptor Example 2 – Read Data

When provided with a list of Classes, Instances and Properties by the vendor, create MD's to read them directly. In this example, data from the 1st instance of class #7's property #24 is read. Specify the correct data type or the 2 byte value field in the response will be decoded incorrectly. The Scan_Interval is not specified so the default is assumed.

```
//    Client Side Map Descriptors

Map_Descriptors
Map_Descriptor_Name,   Data_Array_Name,   Data_Array_Offset,   Function,   Node_Name,   Length,   Cmd,    Class_Type,   Inst_Num,   Prop_Num,   Wattmstr_Data_Type
Read_C7I1P24,          DA_BUIDING01,      109,                 Rdbc,       CPU1,        1,        0x11,   7,            1,          24,         UINT
```

Data is stored in this data array at location 109.

If the length was 2 then property 25 would be read and stored at offset 110. **Ensure that property 25 has the same data type**

Read Property Command.
To Write use 0x12 and change the function to wrbc.

1st instance of class #7's property #24

4.4.6. Map Descriptor Example 3 – Read All Class Data

Use of this command is not recommended unless all the properties of the class being read have the same data type. The Map Descriptor reads data from the 1st instance of class #7. The properties are stored sequentially starting at the offset specified. Ensure the length is sufficient – if there are more items in the response than in the specified length, data items will be discarded.

```
Map_Descriptors
Map_Descriptor_Name,  Data_Array_Name,  Data_Array_Offset,  Function,  Node_Name,  Length,  Cmd,   Class_Type,  Inst_Num,  Wattmstr_Data_Type
Read_C7I1P24,         DA_BUIDING01,     0,                  Rdbc,      CPU1,       100,     0x12,  7,           1,         UINT
```

Ensure length is equal or larger than the number of properties in the response.

Bit	- Decodes as – If (non-zero) store 1 else store 0
Byte	- Decodes as – Store lsb
UINT	-
SINT	-
F.1	- Decodes as – store (word interpreted as uint16 value * 10.0F)
F.2	- Decodes as – store (word interpreted as uint16 value * 100.0F)
F.3	- Decodes as – store (word interpreted as uint16 value * 1000.0F)

5. Configuring the FieldServer as a Wattmaster Server

It is important to note that full Server side functionality is not provided, supported or documented. It has been implemented to satisfy FST's Quality Assurance program and is used to test the Client.

The following information is provided to assist engineers in developing test scripts.

The Server side requires two components

- A text file to define the Database
- Map Descriptors to service commands 0x11 (Read Property), 0x12 (Read All Properties)

Database Definition Files

Unless otherwise specified on the connection the driver assumes the database is kept in a file called 'wattmstr.ini'.
This default can be ovewritten by using the "Simulation_File_Name" connection parameter.

This example defines two classes each with two instances. Each class has 7 properties each of which has a different data type.

```
[GENERAL]
TOTALCLASSES = 2
[CLASS_0]
TYP = 1
NAMELEN = 11
NAME = CLASS_AI_01
TOTALPROPS = 7
PROPNUM_0 = 100
PROPTYP_0 = 0
PROPNAMELEN_0 = 14
PROPNAME_0 = CLASS00_PROP00
PROPNUM_1 = 101
PROPTYP_1 = 1
PROPNAMELEN_1 = 14
PROPNAME_1 = CLASS00_PROP01
PROPNUM_2 = 102
PROPTYP_2 = 2
PROPNAMELEN_2 = 14
PROPNAME_2 = CLASS00_PROP02
PROPNUM_3 = 103
PROPTYP_3 = 3
PROPNAMELEN_3 = 14
PROPNAME_3 = CLASS00_PROP03
PROPNUM_4 = 104
PROPTYP_4 = 4
PROPNAMELEN_4 = 14
PROPNAME_4 = CLASS00_PROP04
PROPNUM_5 = 105
PROPTYP_5 = 5
PROPNAMELEN_5 = 14
PROPNAME_5 = CLASS00_PROP05
PROPNUM_6 = 106
PROPTYP_6 = 6
```

```
PROPNAMELEN_6 = 14
PROPNAME_6 = CLASS00_PROP06
TOTALINSTANCES = 2
INSTNUM_0 = 1000
INSTNAMELEN_0 = 14
INSTNAME_0 = CLASS00_INST00
INSTNUM_1 = 1001
INSTNAMELEN_1 = 14
INSTNAME_1 = CLASS00_INST01
[CLASS_1]
TYP = 1
NAMELEN = 10
NAME = CLASSAI_02
TOTALPROPS = 7
PROPNUM_0 = 100
PROPTYP_0 = 0
PROPNAMELEN_0 = 14
PROPNAME_0 = CLASS01_PROP00
PROPNUM_1 = 101
PROPTYP_1 = 1
PROPNAMELEN_1 = 14
PROPNAME_1 = CLASS01_PROP01
PROPNUM_2 = 102
PROPTYP_2 = 2
PROPNAMELEN_2 = 14
PROPNAME_2 = CLASS01_PROP02
PROPNUM_3 = 103
PROPTYP_3 = 3
PROPNAMELEN_3 = 14
PROPNAME_3 = CLASS01_PROP03
PROPNUM_4 = 104
PROPTYP_4 = 4
PROPNAMELEN_4 = 14
PROPNAME_4 = CLASS01_PROP04
PROPNUM_5 = 105
PROPTYP_5 = 5
PROPNAMELEN_5 = 14
PROPNAME_5 = CLASS01_PROP05
PROPNUM_6 = 106
PROPTYP_6 = 6
PROPNAMELEN_6 = 14
PROPNAME_6 = CLASS01_PROP06
TOTALINSTANCES = 2
INSTNUM_0 = 1002
INSTNAMELEN_0 = 14
INSTNAME_0 = CLASS01_INST00
INSTNUM_1 = 1003
INSTNAMELEN_1 = 14
INSTNAME_1 = CLASS01_INST01
```

```
//=============================================================================
//
// Server Testing
//
//
```

```
// 0.00cC  03 FebC 04 SSS  Created
//
//=================================================================================*/


//=================================================================================
//
// Notes :  None.
//
//
//=================================================================================


//=================================================================================
//
//    Common Information
//

Bridge
Title
Server


//=================================================================================
//
//    Data Arrays
//

Data_Arrays
Data_Array_Name , Data_Format , Data_Array_Length
DA_serv_1       , Float       , 60
DA_serv_2       , Float       , 60
DA_serv_3       , Float       , 60
DA_serv_4       , Float       , 60
DA_serv_5       , Float       , 60
DA_serv_6       , Float       , 60
DA_serv_7       , Float       , 60
DA_serv_8       , Float       , 60


//=================================================================================
//
//    Server Side Connections
//
//
//    Server responds to database polls by using the file wattmstr.ini
//    which contains a database definition. You can change the default
//    file name by specifying the following parameter on the connection: Simulation_File_Name
//
//
//
//

Connections
Port , Baud , Data_Bits , Stop_Bits , Parity , Protocol
P1   , 38400, 8         , 1         , None   , Wattmaster

//=================================================================================
//
//    Server Side Nodes
//

Nodes
Node_Name , Node_ID , Protocol
NODE_01   , 1       , Wattmaster


//=================================================================================
//
//    Server Side Map Descriptors
```

```
//
//
// No MD's are required to respond to commands 0x01 .. 0x06.
//
//
// For each set of properties polled for using command 0x11
// a Server side MD is required that covers the range of propertys requested for a particular
class - instance combo,
// One such MD is required for each Class - instance pair.
//
// For Server MD's the Wattmstr_Data_Type is not used (but might need to be specified as UINT).
// The Server always responds by packing the 2 byte data field with the data extracted from the
DA as a UINT (16 bit)
//
Map_Descriptors
Map_Descriptor_Name  ,Data_Array_Name  ,Data_Array_Offset  ,Node_Name  ,Function  ,Length  ,Cmd
,Class_Type ,Inst_Num,  Wattmstr_Data_Type, Prop_Num
MDSRV_1             ,DA_serv_1       ,0                  ,NODE_01   ,Server   ,100    ,0x11    ,1
,2      , UINT         , 1
MDSRV_2             ,DA_serv_2       ,0                  ,NODE_01   ,Server   ,100    ,0x11    ,2
,2      , UINT         , 1
MDSRV_3             ,DA_serv_3       ,0                  ,NODE_01   ,Server   ,100    ,0x11    ,3
,2      , UINT         , 1
MDSRV_4             ,DA_serv_4       ,0                  ,NODE_01   ,Server   ,100    ,0x11    ,4
,2      , UINT         , 1
MDSRV_5             ,DA_serv_5       ,0                  ,NODE_01   ,Server   ,100    ,0x11    ,5
,2      , UINT         , 1
MDSRV_6             ,DA_serv_6       ,0                  ,NODE_01   ,Server   ,100    ,0x11    ,6
,2      , UINT         , 1
MDSRV_7             ,DA_serv_7       ,0                  ,NODE_01   ,Server   ,100    ,0x11    ,7
,2      , UINT         , 1


// One of the following MD's is required for each class-instance pair to Server responses to
command 0x12
// The length this Map Descriptor determines how many properties are served. There are
// limitations to how many can be served. The protocol spec provides additional info.
Map_Descriptors
Map_Descriptor_Name  ,Data_Array_Name  ,Data_Array_Offset  ,Node_Name  ,Function  ,Length  ,Cmd
,Class_Type ,Inst_Num , Wattmstr_Data_Type ,Prop_Num
MDSRV_8             ,DA_serv_2       ,0                  ,NODE_01   ,Server   ,7      ,0x12    ,1
,1       ,UINT         ,1


// If the Client writes data then a Cmd=0x21 MD is required for each class-instance pair that is
written.
// Ensure that the range of properties being written is covered using the  Prop_Num and Length
parameters.
// This MD uses the Wattmstr_Data_Type to determine how to unpack the value being sent.
Map_Descriptors
Map_Descriptor_Name  ,Data_Array_Name  ,Data_Array_Offset  ,Node_Name  ,Function  ,Length  ,Cmd
,Class_Type ,Inst_Num ,Prop_Num ,Wattmstr_Data_Type
MDSRV_9             ,DA_serv_8       ,0                  ,NODE_01   ,Server   ,60     ,0x21    ,1
,3       ,1        ,UINT


//=================================================================================
//
//    Wattmaster Stats
//

Data_Arrays
Data_Array_Name ,Data_Format ,Data_Array_Length
Wattmstr-stats  ,UINT32      ,500
```

Appendix A. Advanced Topics

Appendix A.1. Auto Create Client

If the ProtoCessor is configured to read the Wattmaster Server's database it can auto-create Client side Map Descriptors to read the data from the Server and Data Arrays for data storage. The auto-creation occurs when the database read is complete.

The Client side Map Descriptors can be created in one of two styles depending on the value of the connection parameter called 'Auto_Config_Client'.

Parameter Value	Style
Yes	The driver creates a Map Descriptor for each *property* of each class instance using command 0x11 (Read Property Value).
Fast	The driver creates a Map Descriptor for each class instance using command 0x12 (Read All Property Values).

When the Client is created the driver creates a file called auto.txt which can be uploaded from the ProtoCessor. This file contains a listing of all the Map Descriptors created.

Naming conventions:

Data Arrays: DA_Cxx_Iyyy where xx is the class number and yyy is the instance number.

Map Descriptors Xxxxxxx[yyy] where xxxxxx is the class name and yyy is the instance number.

Appendix A.2. Auto Create Server

A driver configured to auto-create a Client, can auto-create the BACnet Server side as BACnet-IP, BACnet-Ethernet, BACNet-MSTP, Metasys N2 Open (MN2) or LonWorks. All Map Descriptors created are listed in the auto.txt file which can be uploaded from the ProtoCessor.

The auto-creation process does the following

- Creates a connection using adapter N1
- Creates a node using the Node_ID of the Wattmaster node.
- Creates one Map Descriptor for each data object.
- The object type is based on the property type - See Appendix A.4
- The name of the Map Descriptor and hence the name of the object is formed as follows:
  S_xxxxxxx[yyyyy].zzzzzzzzzzz where
  S_ is a prefix added to all these Map Descriptors

xxxxxxxxx	is the name of the class
Yyy	is the instance number
Zzzzzzzzz	is the name of the property

  Example: MuaII[048].FanInStartUpDelay

Appendix A.3. Server Object Numbering/ID Style

When the Driver autocreates the Server objects for BACnet it numbers them. Numbering can be controlled by specifying the 'Server_Object_ID_Style' parameter on the connection. The highest object ID that a BACnet object may have is approximately 4,000,000. Ensure that the option selected does not produce object ID's exceeding this limit.

Style	Numbering
0 (Default)[3]	The driver numbers objects of each data type sequentially starting at 1. Example AI1, AI2, AI3, AO1, AO2, AO3…etc.
1	The driver uses Wattmaster Class, instance and property number to form the object number using the following formula: Object_ID=Class_Number*100000+Instance_Number*1000+Property_Number Example: 7011001 for class number 7; instance 11; property 1. The object number is not dependent on class.
2	The driver uses Wattmaster Class, instance and property number to form the object number using the following formula: Object_ID=Class_Number*20000+Instance_Number*200+Property_Number Example: 142201 for class number 7 instance 11 property 1. The object number is not dependent on class.
3	The driver uses Wattmaster Instance and property number to form the object number using the following formula: Object_ID = Instance_Number * 1000 + Property_Number Example: 11001 for class number 7 instance 11 property 1. The object number is not dependent on class. This option is suitable for WattMaster devices.

Appendix A.4. Data/Object Type

The Data Type plays two roles

- Tells the driver how to prepare/unpack the two byte data value field in messages which transfer data.
- Is used by the driver when Server side data objects are auto-created by the driver.

Appendix A.1.1. Data Type Selection

When data is transferred between the Wattmaster Server and Client, the value or status of each property is transferred using a 2 byte field. The Wattmstr_Data_Type parameter tells the driver how to interpret this field. In the case of response to polls, this parameter tells the driver how to unpack the field. When the driver sends data, this parameter tells the driver how to prepare the two byte field. **It is easy to see that in a situation where the Client and Server are using different data types that data will be misinterpreted**.

Where the driver is configured to read the database, the driver will discover the class, instances and properties in the database. This information includes the data type of each property discovered. The driver will use this data type, if possible, and thus it isn't

[3] If the Server protocol is Metasys N2 Open (MN2) then this is the only permitted numbering style.

necessary to know the data type in advance. In cases where the driver does not read the database, it is necessary to obtain the data type of each from the vendor.

Appendix A.1.2. Use of Data Type in creating Server Side Objects.

When BACnet Server side objects are created they are allocated a data type according to the following rules. If the Class Type is less than 6 then all properties inherit the type of the class.

Class Type Number	Class Type	Class Property Data Types
1	AI	AI
2	AO	AO
3	AV	AV
4	BI	BI
5	BO	BO
6	BV	BV
7, 8 etc …	User Defined.	Derived from the property type reported by the Wattmaster Server when the database is read. If the Property Data Type is 'Bit' then the object is created as a Binary Object – BI if read only otherwise a BV. All other data types give rise to Analog Objects - AI if read only otherwise AV.

The Wattmaster Server indicates that the data type of a property is read-write by setting the high nibble of the data type to 1.

Appendix A.5. Extra Timeout Control

When a Client poll times out, a timeout is produced which presents as an error. Some customer CPU's are not fast enough to handle every incoming poll correctly. To reduce the number of errors displayed set this parameter to 1. This causes the driver to re-poll once per timeout before the timeout becomes an official error.

Official timeouts contribute to the node status. If there are x official timeouts then the node is considered offline. If there is even 1 official timeout then the driver enters the retry interval which by default is slower than the normal polling rate

Appendix A.6. Command Summary

Additional information can be found in the document “FST Protocol Spec - Wattmaster Advanced Driver Spec (FS-8700-109)”

Command	Action	Notes
0x01	Poll Object Class Count	
0x02	Poll Object Class Configuration	
0x03	Poll Object Class Property	
0x04	Poll Object Instance Count	
0x05	Poll Object Instance Info	
0x06	Configuration Valid / Changed	
0x11	Read Property Values	There are limits to this command. Read the protocol specification document.
0x12	Read All Property Values	There are limits to this command. Read the protocol specification document.
0x21	Write Property Value	

Appendix B. Troubleshooting tips

- Servers typically respond with NAK's when they receive badly formatted polls.
- Servers typically respond with NO_DATA responses when polled for data that doesn't exist.
- Servers typically don't respond at all when the message they receive is so badly formatted that they don't know they have received a full message. During testing of this driver with customer equipment we have noted that CPU's with marginal speed sometimes do not receive complete messages and therefore don't respond at all. For this reason a special method of handling a timeout has been developed for this driver. See Appendix A.5 for more information.
- If the driver has been configured to read a database then it will not process other Map Descriptors until the database read is complete.

Appendix B.1. Driver Error Messages

Some configuration errors produce an error every time a poll is generated. This would fill the error buffer quickly without adding clarity. The driver therefore suppresses subsequent similar messages so that the same error produced by multiple Map Descriptors produces only one error message. Subsequent error messages can be seen on the driver message screen.

Note : In the actual message you will see that %d has been replaced by an integer, %s by text indicating a Data Array or Map Descriptor name and %x by two hex characters.

Error Message	Description and Action Required
WATT:#01 Err. Bad MD Len - defaulting to 1	Map Descriptor length is zero or not set. Driver will assume length as 1.[4]
WATT:#01a Err. Illegal Map Descriptor length, Max 30	If Map Descriptor is defined with Cmd as 0x21 (Set), maximum length is restricted to 30.[4]
WATT:#01b Err. Illegal Map Descriptor length Max 40	If Map Descriptor is defined with Cmd as 0x11 (Get), maximum length is restricted to 40.[4]
WATT:#02 Err. Undefined Cmd	Cmd parameter is blank in configuration file.[4].
WATT:#02a Err. CMD badly formatted	Cmd parameter's value is badly formatted. Note: hex format is 0x?? or 0X??.[4]
WATT:#03 Err. Undefined Class_Type	Class_Type parameter is blank in configuration file.[4]
WATT:#04 Err. Undefined Inst_Num	Inst_Num parameter is blank in configuration file.[4]
WATT:#05 Err. Undefined Prop_Num	Prop_Num parameter is blank in configuration file.[4]
WATT:#06 Err. Undefined Prop_Index	Prop_Index parameter is blank in configuration file.[4].
WATT:#07 Err. Undefined Wattmstr_Data_Type	Wattmstr_Data_Type parameter is blank in configuration file.[4]
WATT:#08 FYI. Database Md %s	The name of the Map Descriptor responsible for getting the database from the Server.[5]
WATT:#11 Err. One Server	When configured to auto create a BACNet Server a BACNet node is

[4] Edit the .CSV file, download the modified file and reset the FieldServer to have the changes take effect.
[5] This message will be printed on Driver message screen.

Error Message	Description and Action Required
Node required.	required in the configuration file.[4]
WATT:#12 Err. Heading not equal to keywords	Call Tech Support.
WATT:#13 Err. MD Name too long <%s>	Map Descriptor indicated cannot be created. Call Tech Support.
WATT:#14 FYI. Re-polling on timeout mode active.	Client re-polls if it gets timeout on first poll. Requires connection is configuration.
WATT:#15 Err. Parse Failed	Client got response in unknown format. Call Tech support with other information displayed with this message.
WATT:#16 Err. Proto Err. Seq Number	Message sequence number for poll-response mis-matched.[5]
WATT:#17 Err. Diagnostic 1	Client set to poll Server with first half of the request only.
WATT:#18 Err. Diagnostic 2	Client set to poll Server with last half of the request only.
WATT:#19 Err. Diagnostic 3	Client set to do a poll proceeded with garbage characters.
WATT:#21a Err. Proto Err. Bad CMD code. Poll=%x Resp=%x	Command did not match while polling for database.
WATT:#21b Err. Proto Err. Bad CMD code. Poll=%x Resp=%x	Command did not match while polling for data other than database.
WATT:#21c Err. Proto Err. Bad CMD code. Poll=%x	Client made poll with unknown command.
WATT:#21d FYI. Unknown CMD %d (%2X)	Client trying to poll with unknown command.
WATT:#22 Err. Cmd0x5 Null Ptr.	Take a log and call Tech Support
WATT:#23 Err. Device reports none of polled properties exist. MD = %s	Server device retuned none of the requested properties.
WATT:#24 FYI. Confg DataBase Complete	Client received the entire configuration database.
WATT:#25 FYI. Unknown Confg Change Code %d ,assuming 1	Returned code for configuration change is non zero but not 1 - driver considers it as 1 and will proceed to get new database.
WATT:#26 Err. Cant store at %u da %s dalen %u	There is insufficient space to store in indicated Data Array.
WATT:#27a FYI. MD's <%s> bad Write Status	Response shows at least one of the properties is not written successfully.
WATT:#27b FYI. %d th Property written status is 1	Indicates very first property which did not writte successfully.[5]
WATT:#28 Err. Unknown Md CMD %d	Map Descriptor is asked to store data with unknown command.
WATT:#31 Err. Asked to calc chk. 1st byte Exist/Reqd %2X/%2X	Found illegal start byte while calculating checksum.[5]
WATT:#32 FYI. You could use an Array called <%s> to expose diagnostic info	Declare indicated Data Array in Data_Array section for driver to expose diagnostic statistics or to force driver to follow diagnostic commands. See Appendix B.2
WATT:#33 FYI. You could use an Array called <%s> to dump database info	Declare indicated Data Array in Data_Array section if you want driver to dump database info in Data Array.
WATT:#34 FYI. Waiting for	Driver is waiting for permission to dump database info in a file. Set

Error Message	Description and Action Required
permission to dump DB.	offset WATT_OK_TO_DUMP to 1 in the stats array.
WATT:#34a ERR: Cannot create dump file %s	Internal problem to create a file.
WATT:#34b ERR: Cannot open file %s	Internal problem to open a file.
WATT:#35 FYI. Dump Da insufficient space	Insufficient space in Data Array declared to store database information. Information will be truncated.
WATT:#36a FYI. Unknown class type. Treat as 'AV'	
WATT:#36b FYI. Unknown class type. Treat as 'AV'	
WATT:#36c FYI. Unknown class type. Treat as 'AV'	
WATT:#37 Err. No Memory at FieldServer	Database is too big to be handled by FieldServer. Call Tech Support.
WATT:#41 FYI. Response inhibited.	Offset WATT_STAT_RESPONSE_INHIBITED has been set to 1 in the stats array. Change to zero to remove this restriction.
WATT:#42 Err Diagnostic 4	Server set to respond with incorrect checksum.
WATT:#43 FYI Server_Object_ID_Style overridden to 'Default'	You may ignore this error if you are satisfied that default style is appropriate. Read more in section Appendix A.3 To eliminate this error, edit the config and change the value of the parameter to 'Default' or '0'. Download the correct config file and restart the FieldServer.
WATT:#46 Err. Class has %d properties But is serving info for property #%d"	When the server reported its class information it reported the property count as x. Now, when polled for property attributes (using cmd=0x03), it is reporting the attributes of a property whose index is greater than x. If you are using the FieldServer as a server then report this to tech support. If you are using a Wattmaster or other Vendor device then report this error to the vendor.

Appendix B.2. Driver stats

In addition to the standard FieldServer operating statistics the driver exposes certain key stats in a Data Array if required. An upstream device can then monitor these stats.

Add the following to your configuration file to activate these stats.

```
// Expose Driver Operating Stats.

Data_Arrays
Data_Array_Name,                      Data_Format,        Data_Array_Length
WATT-stats,                           UINT32,             200
```

Length must be sufficient to store error messages.

Table shows offset (= Stat Number) where stat will be stored in "WATT-stats" Data Array.

Stat #	Stats	Description
0	WATT_CMPLTDATABASE	1 when Client has completed latest database. 0 if latest database not completed or being captured.
1	WATT_STATUSDATABASE	Database status. 1= changed, 0= unchanged. Server will return 0 after responding once with status 1.
2	WATT_OK_TO_POLL	During field testing set to 1 to poll. During home testing with FieldServer set to 0 to poll.
3	WATT_OK_TO_DUMP	Set to 1 to print database info to file "Wattmaster _db.txt".
4	WATT_STAT_NODE_CREATED _BACNET_IP	Number of BacNet_IP Nodes created in auto Server mode.
5	WATT_STAT_NODE_CREATED _BACNET_ETH	Number of BacNet_ETH Nodes created in auto Server mode.
6	WATT_STAT_RESPONSE_INHIBITED	To stop Server response, set to 1.
7	WATT_STAT_RESEND_OF_POLL	Number of times Client re-polls the node.
8		Obj Id of 1st bacnet object created is stored here
9		Obj Id of 2nd bacnet object created is stored here
10		Obj Id of 3rd bacnet object created is stored here
11		Obj Id of 4th bacnet object created is stored here
12		Obj Id of 5th bacnet object created is stored here
13		Obj type of 1st bacnet object created is stored here
14		Obj type of 2nd bacnet object created is stored here
15		Obj type of 3rd bacnet object created is stored here
16		Obj type of 4th bacnet object created is stored here
17		Obj type of 5th bacnet object created is stored here
18		When set to 1 the Server side will respond reporting all properties as read only
19		Driver sets this to 1 to report that a BACnet MSTP node has been created.
20		Set to 1 to tell driver to print message each time it stores data.
21		Driver sets this to 1 to report that a Metasys node has been created.
22		Count of the number of Server objects made whose MD data type = BI
23		Count of the number of Server objects made whose MD data type = BO
24		Count of the number of Server objects made whose MD data type = BV
25		Count of the number of Server objects made whose MD data type = AI
26		Count of the number of Server objects made whose MD data type = AO
27		Count of the number of Server objects made whose MD data type = AV

Appendix C. Protocol Specification

The Wattmaster Serial Protocol defines the interface protocol between the Host CPU and the FieldServer's ProtoCessor. The Wattmaster Protocol allows the Host CPU to provide data information to the ProtoCessor. The Host CPU is always the Server and the ProtoCessor is always the Client. The ProtoCessor requests information from the Host CPU according to the Wattmaster Protocol and then converts the information to other network protocols. The Host CPU will not send unsolicited messages to the ProtoCessor.

Appendix C.1. Protocol Definition

Appendix A.1.3. Physical Link

Physical Link: 5 Volt TTL signal only
Baud Rate: 38400
Serial Protocol: 10 bits with 1 start bit, 8 data bits, and 1 stop bit, (N,8,1)
Flow of Data: Full Duplex

Appendix A.1.4. Data Order

The Data Order of all two byte values within the Packet Message are formatted as Big-Edian order which defined as Most Significant Byte First.

Appendix A.1.5. DLE Insertion

This was considered and rejected for the following reasons:

- No more than a 2″ comm link will exist between Host CPU and the Protocessor which makes the risk of errors very, very small.
- The presence of an STX and a checksum will allow resync of the messages eventually.

Appendix C.2. Packet Format

Appendix A.1.6. The POLL / RESPONSE Packet

PA	SZ	CMD	MN	……MSG……	SUM

PA:	Packet Preamble, 0x02 (STX). Each packet must begin with this preamble (or PB) to indicate the beginning of one packet.
PB	Alternate Preamble, for Piggy Back messages. Value: 0x04.
SZ:	Packet Size, this is counted from the CMD to the end of the MSG. The maximum this can be is 253 characters. This allows for a maximum message length of 256 characters, which is a limitation on some MCU page sizes. This byte must immediately follow the one byte preamble.
CMD:	Packet Command. Refer to Appendix A.6 for a table of different commands (CMD).
MN:	Packet Message Number, can be 0...255. This message number is a one byte value that serves as a flow management device for the package request source.
MSG:	Packet Message, this data depends on the Packet Command.

SUM:	Packet Checksum, a rotate left and XOR of the message from PA to all the end of the MSG.
	
	The checksum is calculated by rotating the original checksum to the left and then doing an XOR with the new byte. The starting checksum is zero. This simple C function shows an example of this checksum calculation.
	
	`unsigned char CheckSum;`
	
	`CheckSum = ( (CheckSum<< 1 ) // rotate CheckSum Left`
	`| (CheckSum>>7) ) // retain the leftmost bit`
	`^ NewByte; // combine with new byte`

Appendix A.1.7. ACK

PA	SZ	ACK	MN	SUM

SZ = 1
CMD = ACK = 0xFE
SUM = 0xXX

Appendix A.1.8. NAK

A NAK is transmitted when the Host CPU cannot interpret the poll because of a checksum, bad format (no preamble etc), unknown command, bad length etc.

PA	SZ	NAK	MN	ERR	SUM

CMD = NAK = 0xFF
Refer to Appendix B.1 for a list of Error (ERR) Codes.

Appendix C.3. Packet Flow

A transaction comprises a poll from the ProtoCessor and a response from the Host CPU. Only one transaction can be pending at a time. A transaction is deemed to have timed out if the Host does not respond within a specified time period. This time period is set by default to 2 seconds.

Once a transaction is complete the ProtoCessor may begin the next transaction. This means that the next poll could occur immediately, and the host CPU receive buffer needs to be ready to receive the next characters.

Appendix C.4. How Data is organized in the Host CPU

The ProtoCessor will request the Information Setup Configuration from the Host CPU during power up. This defines what information is available from the Host CPU to the ProtoCessor. The Information Setup Configuration is based on the data object model concept.

Data is organized into Data Object Classes. A Host CPU may have a variable number of Data Object Classes.

Data Object Class Types

01:	AI		05:	BO
02:	AO		06:	BV
03:	AV		07…	User Defined
04:	BI			

The Host CPU has a series of Data Objects. Each Data Object is an instance of the Data Object Class. Thus all the attributes of the Class are inherited by the instance and each Data Object is uniquely identified by its Class and its instance number.

Example:

AI:1 (Instance 1 of the Data Object Class whose type is AI)
AI:2 (Instance 3 of the Data Object Class whose type is AI)
AI:3 (Instance 4 of the Data Object Class whose type is AI)
BI:1 (Instance 1 of the Data Object Class whose type is BI)

Each Data Object Class contains an array of properties which are also inherited by each Data Object.

Each property is defined by a property index, a property name, a property data type and a value. While every Data Object instance has the same list of properties as the Data Object Class that it was derived from, the value of a property belongs only to the instance and not the class. The index is used to reference a particular property in poll/response. The property data type is used to interpret the value field when the property value is polled for.

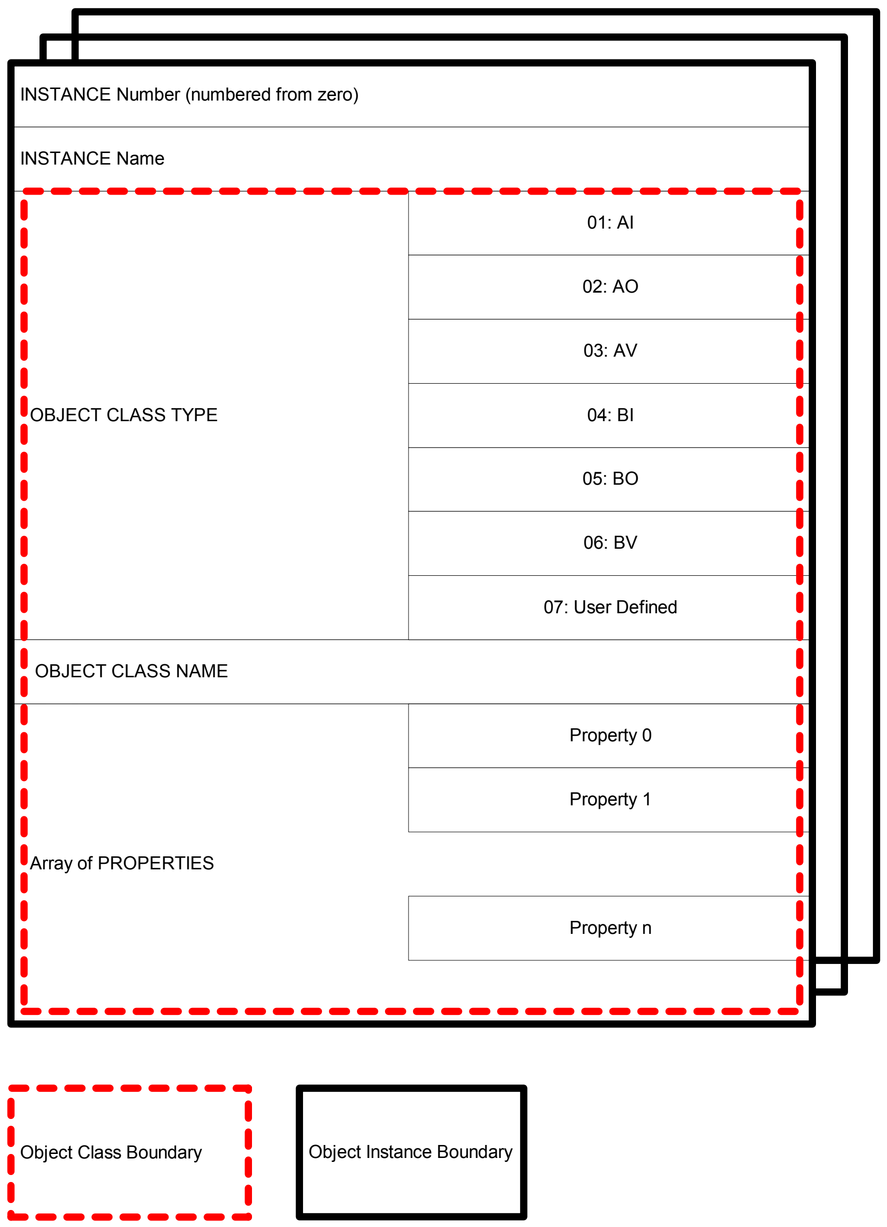
Example

The class called AI may have two properties; Current Value and Average Value.
An instance of AI is created to store the measured room temp. The Instance is named 'Temperature'
An instance of AI is created to store the measured room humidity. The Instance is named 'Humidity'

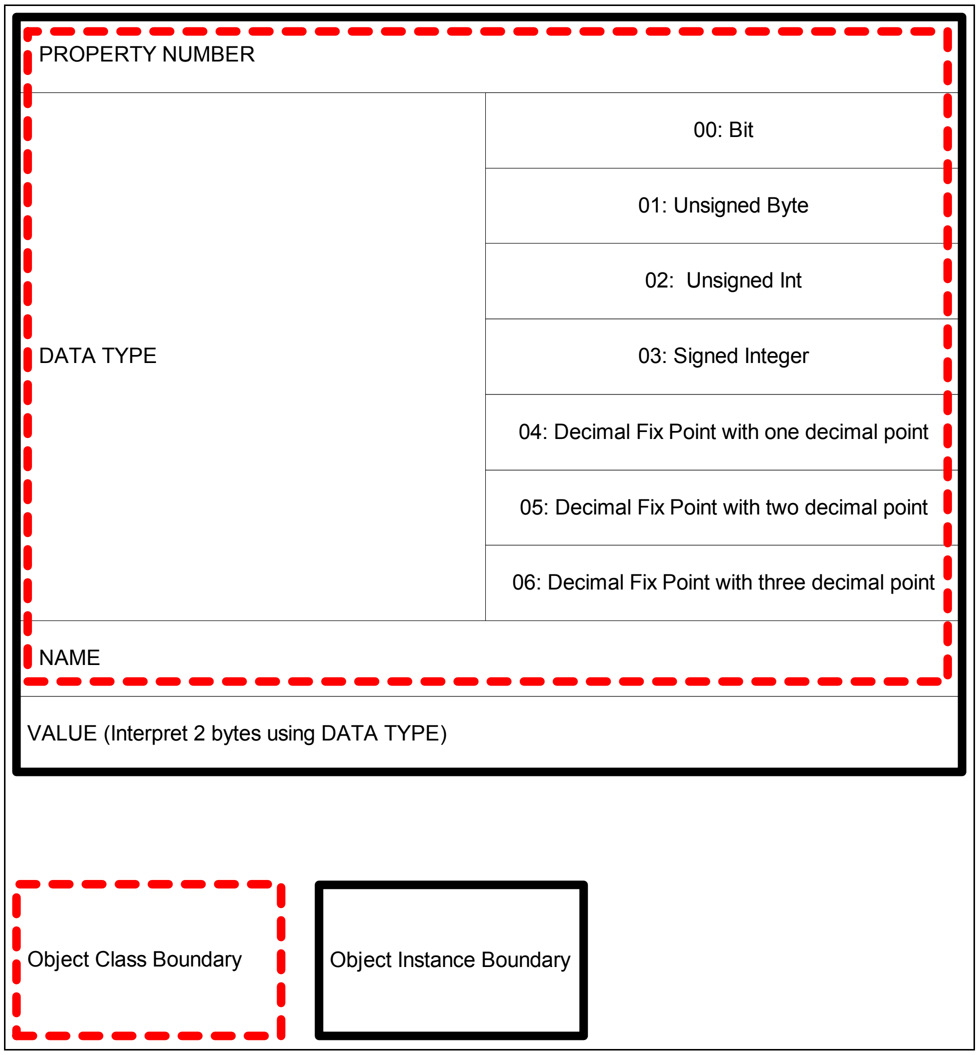
Each instance has both properties. Thus it is possible to read the value of the Current Value property of the Temperature Instance and of the Humidity instance.

Class Type(:instance #)	AI	AI:1	AI:2
Instance Number	Not Defined.	1	2
Name	Analog Input	Temperature	Humidity
Property 1 Name	Current Value	Current Temp.	Current Humidity
Property 1 Value	Not Defined	68	50
Property 2 Name	Average Value	Avg Temp	Avg Humid
Property 2 Value	Not Defined	68	50

Appendix A.1.9. Figure: Data Object Classes and Instances



Appendix A.1.10. Figure: Data Properties


PROPERTY NUMBER
DATA TYPE
00: Bit
01: Unsigned Byte
02: Unsigned Int
03: Signed Integer
04: Decimal Fix Point with one decimal point
05: Decimal Fix Point with two decimal point
06: Decimal Fix Point with three decimal point
NAME
VALUE (Interpret 2 bytes using DATA TYPE)
Object Class Boundary
Object Instance Boundary

Appendix A.1.11. Data Type

Data Types are defined as:

00:	Bit value
01:	Unsigned Byte value
02:	Unsigned Integer value
03:	Signed Integer value
04:	Decimal Fix Point value with one decimal point
05:	Decimal Fix Point value with two decimal point
06:	Decimal Fix Point value with three decimal point

Appendix A.1.12. Property ID

Each Property should be identified as a Property ID. The Property ID is defined as:

OBJECT CLASS	OBJECT INSTANCE NUMBER	PROPERTY NUMBER

The total bytes of the Property ID is determined by using the Field Definitions in Appendix C.

Appendix C.5. Packet Command

See Appendix A.6 for a table of commands. Examples of the responses to commands follow below:

Command	Poll from ProtoCessor	Possible Response from Host System
01	01	ACK,NAK
02	02	Response, NAK, NO DATA
03	03	Response, NAK, NO DATA
04	04	Response, NAK,
05	05	Response, NAK, NO DATA
06	06	Response, NAK,
11	11	Response, NAK, NO DATA
12	12	Response, NAK, NO DATA
21	21	Response, NAK, NO DATA

Sequence

- Client polls for the number of object classes.
- Then starting at index 0 it polls for the **name and type of each class,** incrementing the index by 1 until a NO_DATA response is obtained indicating the end of list.
- Then starting at class index 0 and working through the list of classes it polls for the **properties** of each class incrementing the index by 1 until a NO_DATA response is obtained indicating the end of list.
- Thenit pollsl for the **number of instances** of each class incrementing the index by 1 until a NO_DATA response is obtained indicating the end of list.
- Finally it polls for the **instance name and number** of each instance of each class incrementing the index by 1 until a NO_DATA response is obtained indicating the end of list.
- It is now possible to poll for property values for each property of each instance using command 0x11 and 0x12. It is also possible to set a property value by using command 0x21.

Appendix D. Commands

Appendix D.1. Command 0x01 : Poll Object Class Count

The ProtoCessor uses this poll to learn how many object classes exist in the Host CPU. This packet contains no message data. The packet details as:

Poll from ProtoCessor:

03	04	01	MN	SUM

The Host CPU should respond with ACK package as:

03	06	FE	MN	TOTAL CLASSES	SUM

Appendix D.2. Command 0x02 : Poll Object Class Configuration

The ProtoCessor uses this message to obtain a list of all Object Classes. Poll for Object Class with index 0. Response contains Object Class Info or a NAK. A NAK indicates the Object with specified index does not exist. ProtoCessor must start polling with Index 0 and proceed, incrementing the index each poll until a NO_DATA response is obtained indicating the end of the Object Class list. It is valid to poll for Object Class configuration for any index at any time – if used non-sequentially the response from this poll cannot be used to infer the total number of Object Classes.

Once this command is complete the Poll Object Configuration Packet should be sent from ProtoCessor to Host CPU. The packet details as:

Poll from ProtoCessor:

Packet

03	SZ	02	MN	MSG	SUM

MSG

INDEX

Response from Host CPU:

03	SZ	02	MN	OBJ CLAS (Type)	TOTAL PROPERTY	STR LEN	NAME STR	SUM

Where
STR LEN = the length of the name string;
NAME STRING = the Object Class Name String.

Appendix D.3. Command 0x03 : Poll Object Class Property

Once the ProtoCessor has a list of Object Classes it can poll for the properties of each class. It polls for property index zero of each class. If the property exists then the property information is sent in a response, the ProtoCessor increases the index and polls for the next property. If a NO_DATA response is obtained then the list is considered to be complete.

Using the commands (0x02 and 0X03) the ProtoCessor will generate a complete list of Data Object Classes and the properties that form each class.

Poll from ProtoCessor:
Packet

03	SZ	03	MN	MSG	SUM

MSG

OBJ CLASS INDEX	INDEX

Where
OBJ CLASS INDEX = Index of the Object Class obtained using command 0x02.
INDEX = 2 Byte property index value.

Response from Host CPU:
NAK, NO_DATA or the following message

03	SZ	03	MN	OBJ CLAS	PROPERTY NUMBER	DATA TYP	STR LEN	NAME STR	SUM

Where
STR LEN = The length of the property name string
NAME STRING = The Property's Name String.

Appendix D.4. Command 0x04 : Poll Object Instance Count

The results of this poll indicate the number of instances of a specific Data Object Class.

Poll from ProtoCessor:
Packet

03	SZ	04	MN	MSG	SUM

MSG

OBJ CLASS INDEX

Where
OBJ CLASS INDEX = Index of the Object Class obtained using command 0x02.

Response from Host CPU:
NAK, NO_DATA or the following message

03	06	04	MN	TOTAL INSTANCES	SUM

Appendix D.5. Command 0x05 : Poll Object Instance Info

The ProtoCessor uses this command to learn the name (and possibly the count) of each instance of each Data Object Class.

The instance name is used to name the data objects in the FieldServer. Consider the case where the Wattmaster Protocol is used to poll a device for its data configuration and then make the data objects available to a remote Client using BACNet for example. The instance name is used to name these BACnet objects. In cases where the instance name is NULL or spaces, the ProtoCessor will use the object class name to name the instances. Thus if an object class is named 'Temperature', the first instance of the class will be named 'Temperature[1]', the 2nd will be name 'Temperature[2]' etc.

Using this command the Client (ProtoCessor) learns the name and the instance number of each instance. The instance number is used to poll for property values.

Poll from ProtoCessor:
Packet

03	SZ	04	MN	MSG	SUM

MSG

OBJ CLASS INDEX	INSTANCE INDEX

Response from Host CPU:
NAK, NO_DATA or the following message

03	SZ	05	MN	INSTANCE NUMBER	STR LEN	NAME STR	SUM

Where
STR LEN = the length of the Data Object Instance Name String
NAME STRING = the Data Object Instance Name String

Appendix D.6. Command 0x06 : Configuration Valid / Changed

It is possible that the HOST CPU is reconfigured so that its list of object types, object instances changes. The ProtoCessor sends this poll to learn if the configuration has changed. If it has the ProtoCessor should re-build its database.

The Host CPU sets the code to zero once it has responded to the poll.

Poll from ProtoCessor:
Packet

03	SZ	06	MN	SUM

Response from Host CPU:
NAK, NO_DATA or the following message

03	SZ	06	MN	CODE	SUM

Code may have one of the following values
0 : No change
1: Changed

Appendix D.7. Command 0x11 : Read Property Values

This poll is used by the ProtoCessor to read one or more real time data values from one or more Data Objects. The Read Property Value Packet should be sent from the ProtoCessor to the Host CPU for requesting information from the host system. The packet details as:

Poll from ProtoCessor:

03	SZ	11	MN	TOTAL	ARRAY OF ID	SUM

Where a single ID =

OBJ CLASS INDEX	OBJECT INSTANCE NUMBER	PROPERTY NUMBER

Where
OBJECT INSTANCE NUMBER is not an INDEX. This value is learned using command 0x05.
PROPERTY NUMBER is not an INDEX. This is number learned using command 0x03.

The first byte of the Packet Message should be the Total Number of Properties Requested. The Total Number should not exceed 40. Following the Total byte, is an array of Property ID's to be read.

Response from Host CPU:
NAK, NO_DATA or the following message

03	SZ	11	MN	TOTAL	ARRAY OF ITEM INDEX & VALUE	SUM

The first byte of the return message should be the Total number of values returned. The Total number of values returned **may not be the same as** the Total number of properties requested because non-existent property ID's would be omitted from the return packet. Following the Total bytes should be an array of Index Number and Property Value. Each returned value will be preceded with an Index number corresponding to the Index of the Property ID within the Property ID array of the request packet. If all properties requested are available, the Index number should be 0, 1, 2,,, Total-1. If there are Property ID's not available, the Index of those Property ID's will be skipped from the return packet.

If none of the Properties requested are available, the Host CPU will return with NO_DATA packet

Appendix D.8. Command 0x12: Read All Property Values

The command is used by the ProtoCessor to read the values of all the properties of a single instance of a Data Object.
The Read All Property Packet should be sent from the ProtoCessor to the Host CPU for requesting all properties of the specified object from the Host CPU. The packet details as:

Poll from ProtoCessor:

03	SZ	12	MN	OBJECT CLASS	OBJECT INSTANCE NUMBER	SUM

The first two bytes of the Packet Message should be the Object Type number of the Object to read. The following byte should be the Object Instance Number of the Object to read.

Response from Host CPU:
NAK, NO_DATA or the following message

03	SZ	12	MN	TOTAL	NEXT	ARRAY OF PROPERTY NUMBER & VALUE	SUM

The first byte of the return message should be the Total number of Property Values returned. If the following NEXT byte is not zero it indicates that this packet cannot hold all the property values and more return packets will be sent to the ProtoCessor with the same message number. The following bytes are arrays of Property Number and Property Values. The Property Number precedes the Property Value and identifies which property the value is corresponding to.

If the data object doesn't exist or if the object has no properties then the Host CPU will return with NO_DATA packet

Appendix D.9. Command 0x21: Write Property

The Write Property Packet should be sent from the ProtoCessor to the Host CPU for writing setpoints to the Host CPU. The packet details as:

Poll from ProtoCessor:

03	SZ	21	MN	TOTAL	ARRAY OF PROPERTY ID AND VALUE	SUM

The first byte of the Packet Message should be the Total Number of Properties to be written. Following the Total byte, is an array of Property ID's and the value for the targeted Property ID to be written.

Where a single ID & Value =

OBJECT CLASS	OBJECT INSTANCE NUMBER	PROPERTY NUMBER	VALUE

Where
OBJECT INSTANCE NUMBER is not an INDEX. This value is obtained using command 0x05.
PROPERTY Number is not an INDEX. This value is obtained using command 0x03.

Response from Host CPU:
NAK, NO_DATA or the following message

03	SZ	21	MN	TOTAL	ARRAY OF WRITTEN STATUS	SUM

The TOTAL field should be the Total number – the same as the requested packet. Following the Total field should be an array of Written Status defined as follows:

00:	Value was written normally.
01:	Property ID is not available.
02:	Property is read only property.
03:	Written value is out of range.
04:	Other illegal status.

Max String length = 256.

If the Data Object Class does not exist, has no instances, the requested instance doesn't exist or if the requested instance has no properties then the NO_DATA response is sent by the host CPU.

Appendix E. Data Tables

Appendix E.1. Message Fields

Field Name	Size (Bytes)	Description
Object class Object class type	2	A numeric value used to indicate the class type. See chapter Appendix C.4 for valid values.
Index Obj class index Instance index Item index	2	A value used to index a particular instance of an object class or property. Instances numbers begin at zero: that is, the first item in a list has an index of zero.
Instance number	2	Differs from an index number. The instance number is used for polling (read/write) property values. For those commands the instance number is used. To learn the relationship between instance numbers and instance indices command 0x05 is used.
Total objects	2	
Obj str len	1	Reports the number of bytes that constitute a string. Property names, object class names and instance names are strings.
Name string	Variable. Max=?	
Data typ	1	See chapter 0
Code	1	
Object instance number	Same as index	
Property number	Same as index	
Value	2	Interpret as per chapter 0
Next	1	
Written status	1	
Total instances	2	A count of the number of instances
Total classes	2	A count of the number of object classes
Total	1	Total number of properties read or to write or written

Appendix E.2. Command List

Table of possible commands (CMD) for Wattmaster Serial Protocol messages.

Poll From ProtoCessor	
01	Poll Object Class Count
02	Poll Object Class Configuration
03	Poll Object Class Property
04	Poll Object Instance Count
05	Poll Object Instance Info
06	Configuration Valid / Changed
11	Read Property Values
12	Read All Property Values
21	Write Property Values
E0	Loop Back Test
E1	Supply Protocol Version
Response from Host CPU	
FD	NO DATA
FE	ACK
FF	NAK
05	End of Configuration (EOC)
06	Delete Object Instance (DOI)

Appendix E.3. Error Codes

Table of Error Codes (ERR) for Wattmaster Serial Protocol messages.

Error Codes from Host CPU	
01	
02	Bad Command
03	Unidentified Property
04	Bad Object Index
05	Bad Instance Index
06	Bad Property Index
07	
08	Data Not Ready (Startup Condition)

THIS PAGE INTENTIONALLY LEFT BLANK