Driver Manual

FS-8704-26 SNMP

Version1 & Version2

APPLICABILITY & EFFECTIVITY
Effective for all systems manufactured after May 2021.

Driver Revision: 1.10
Document Revision: 2.B

MSA Safety
1991 Tarob Court
Milpitas, CA 95035
Website: www.MSAsafety.com

U.S. Support Information:
+1 408 964-4443
+1 800 727-4377
Email: smc-support@msasafety.com

EMEA Support Information:
+31 33 808 0590
Email: smc-support.emea@msasafety.com

Contents

1   Description

The SNMP-STD driver allows the FieldServer to transfer data to and from devices over Ethernet using the SNMP Version V1 or V2c protocol. The FieldServer can emulate a Server (SNMP Agent) or Client (NMS – Network Management Station).

The FieldServer provides a generic MIB (Management Information Base) file that sets out the OID (Object Identifiers) structure. The FieldServer Enterprise ID is 6347. A selection of standard MIB-2 OID's are supported to allow interaction with popular Network Management packages.

When configured as an SNMP Agent (Server) the SNMP-STD driver allows SNMP Get, GetNext (walk) and Set commands to access Data Arrays using the Integer type. The SNMP v1 protocol does not make provision for Floats.

The SNMP-STD driver can send SNMP traps. The structure for SNMP Traps is provided in the FieldServer's generic MIB file.

The FieldServer also support custom MIBs. It supports setting a custom enterprise ID, object names and custom traps or informs. In custom configurations, the FieldServer supports various data types as specified in **Section 6.4**.

When configured as a Client, the FieldServer can read objects from the Server using Get, GetNext(walk) or GetBulk commands. The GetBulk command is very useful to transfer large amounts of data. The FieldServer can update objects in Agent using the Set command.

The FieldServer can accept any trap or inform as long as all the objects in the message are encoded with a full OID.

The Client side of the driver is considered as a custom configuration.

FieldServer Mode	Nodes	Comments
Maximum remote Nodes the driver Client can connect to	100 (typical max recommendation)	Depends on update rate, system resources, network health, server response time etc.
Maximum remote Nodes that can connect to the Driver Server	100 (typical max recommendation for poll response configuration and only 1 can also be a trap receiver)	Depends on update rate, system resources and network health etc.
Maximum local Client nodes the driver can emulate on one platform	100 (typical max recommendation)	Depends on update rate, system resources, network health, server response time etc.
Maximum local Server Nodes the Driver can emulate on one platform	254	Virtual IP Addresses
Capable of Emulating local Server and Client at the same time?	No	

2   Driver Scope of Supply

2.1   Supplied by MSA Safety

Part #	Description
FS-8915-10	UTP cable (7 foot) for Ethernet connection

3　Hardware Connections

The FieldServer is connected to the Ethernet using the UTP cable supplied. A typical hardware configuration is shown below:

FieldServer as an SNMP Agent (Server)

Network Management Station

FieldServer SNMP Agent/ Server

TCP/IP Cloud

3rd Party Nodes (Non-SNMP)
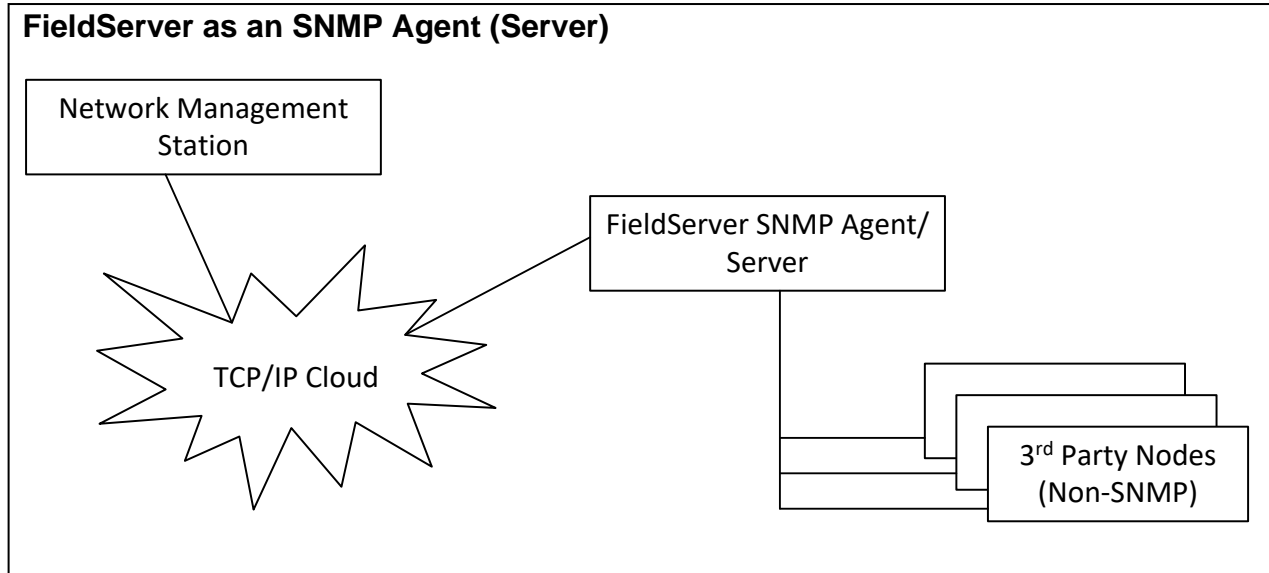
3.1　Hardware Connection Tips / Hints

- Ensure that the FieldServer and all Nodes to be monitored via SNMP have the same Netmask setting.

- Default IP ports
  - Port 161 – Poll/Walk commands
  - Port 162 – Traps

4   Data Array Parameters

Data Arrays are "protocol neutral" data buffers for storage of data to be passed between protocols. It is necessary to declare the data format of each of the Data Arrays to facilitate correct storage of the relevant data.

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, Byte, Uint16, Uint32, Sint16, Sint32
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.	1-10000

Example

```
//   Data Arrays

Data_Arrays
Data_Array_Name     , Data_Format   , Data_Array_Length
G_ALM_STAT_1        , Uint16        , 7
G_ALM_CNTR_1        , Uint16        , 6
BRN_CUR_ALM_1       , Uint16        , 253
LOCATION_1          , Byte          , 128
AUX_CNT64_          , Uint32        , 2
DA_CONFIG           , Bit           , 1
```

4.1.1   Standard Configuration

A special Data Array naming convention is used to map FieldServer Data Arrays into the SNMP OID addressing scheme. Any Data Arrays that are to be visible via SNMP have to be named in the following way:

```
Data_Arrays
Data_Array_Name     , Data_Format   , Data_Array_Length
// Scalar Data Arrays
SNMP_AI             , AI            , 21
SNMP_AO             , AO            , 21
SNMP_AV             , AV            , 21
SNMP_BI             , BI            , 21
SNMP_BO             , BO            , 21
SNMP_BV             , BV            , 21
//Vector Data Arrays
SNMP_SV_1           , BV            , 21
SNMP_AV_1           , AV            , 21
SNMP_BV_1           , BV            , 21
SNMP_SV_2           , BV            , 21
SNMP_AV_2           , AV            , 21
SNMP_BV_2           , BV            , 21
```

A Scalar data array should be used to represent data of the same type (e.g.: Analog values) in a single logical SNMP block. Vector data arrays should be used only if it is required to see data of the same type in multiple logical SNMP blocks. For example, suppose the FieldServer has to represent data (1000 Analog values in this example) from multiple sources (ID's 10 and 11 in this example).

To do this, one would declare vector Data Arrays SNMP_AV_10 & SNMP_AV_11, where 10 and 11 are identifiers (vector numbers) to represent source ID's.

SNMP OIDs for vector data arrays (assuming 500 in each array) will be: 1.3.6.1.4.1.6347.2.52.1.1.10.1 - 1.3.6.1.4.1.6347.2.52.1.1.10.500 and
1.3.6.1.4.1.6347.2.52.1.1.11.1 - 1.3.6.1.4.1.6347.2.52.1.1.11.500

If one Scalar Array (SNMP_AV) had been used instead, the OID's would simply have been: 1.3.6.1.4.1.6347.2.4.1.1.1 - 1.3.6.1.4.1.6347.2.4.1.1.1000

The Data_Format[1] and Data_Array_Length may be freely chosen. The Scalar Data Array needs to be determined as specified in the example above. The Vector Data Array name must be in SNMP_xV_y format, where x is the vector type (S-String, A-Analog, B-Binary) and y is the Integer number representing the vector number. Data will be mapped from Data Array Offset 1, e.g. OID 1 will be mapped to the specified Data Array at Offset 1, OID 3 will be mapped at Offset 3.

FieldServer OID's are based upon the Data Array name and length. The following table shows OID ranges for the Data Arrays declared in the previous table.

Scalar Data_Array_Name, Data_Format, Data_Array_Length	Value OID range	Description OID range
SNMP_AI, AI, 21	1.3.6.1.4.1.6347.2.2.1.1.1-1.3.6.1.4.1.6347.2.2.1.1.20	1.3.6.1.4.1.6347.2.2.1.2.1-1.3.6.1.4.1.6347.2.2.1.2.20
SNMP_AO, AO, 21	1.3.6.1.4.1.6347.2.3.1.1.1-1.3.6.1.4.1.6347.2.3.1.1.20	1.3.6.1.4.1.6347.2.3.1.2.1-1.3.6.1.4.1.6347.2.3.1.2.20
SNMP_AV, AV, 21	1.3.6.1.4.1.6347.2.4.1.1.1-1.3.6.1.4.1.6347.2.4.1.1.20	1.3.6.1.4.1.6347.2.4.1.2.1-1.3.6.1.4.1.6347.2.4.1.2.20
SNMP_BI, BI, 21	1.3.6.1.4.1.6347.2.5.1.1.1-1.3.6.1.4.1.6347.2.5.1.1.20	1.3.6.1.4.1.6347.2.5.1.2.1-1.3.6.1.4.1.6347.2.5.1.2.20
SNMP_BO, BO, 21	1.3.6.1.4.1.6347.2.6.1.1.1-1.3.6.1.4.1.6347.2.6.1.1.20	1.3.6.1.4.1.6347.2.6.1.2.1-1.3.6.1.4.1.6347.2.6.1.2.20
SNMP_BV, BV, 21	1.3.6.1.4.1.6347.2.7.1.1.1-1.3.6.1.4.1.6347.2.7.1.1.20	1.3.6.1.4.1.6347.2.7.1.2.1-1.3.6.1.4.1.6347.2.7.1.2.20

---

[1] The SNMP driver server transfers all data array values to requesting clients as signed integers or SInt32, except for ASCII characters which exchange in strings. The Data_Format for data arrays may still be freely chosen but be aware that value truncation and a change of sign of driver displayed values may take place when these values are transferred to the client.

Vector Data_Array_Name, Data_Format, Data_Array_Length	Value OID range	Vector Description OID
SNMP_SV_1, BV , 21	1.3.6.1.4.1.6347.2.51.1.1.1.1- 1.3.6.1.4.1.6347.2.51.1.1.1.20	1.3.6.1.4.1.6347.2.51.1.2.1
SNMP_SV_2 , BV, 21	1.3.6.1.4.1.6347.2.51.1.1.2.1- 1.3.6.1.4.1.6347.2.51.1.1.2.20	1.3.6.1.4.1.6347.2.51.1.2.2
SNMP_SV_x , BV , y	1.3.6.1.4.1.6347.2.51.1.1.x.1- 1.3.6.1.4.1.6347.2.51.1.1.x.(y-1)	1.3.6.1.4.1.6347.2.51.1.2.x
SNMP_AV_1, AV, 21	1.3.6.1.4.1.6347.2.52.1.1.1.1- 1.3.6.1.4.1.6347.2.52.1.1.1.20	1.3.6.1.4.1.6347.2.52.1.2.1.1 - 1.3.6.1.4.1.6347.2.52.1.2.1.20
SNMP_AV_2, AV, 21	1.3.6.1.4.1.6347.2.52.1.1.2.1- 1.3.6.1.4.1.6347.2.52.1.1.2.20	1.3.6.1.4.1.6347.2.52.1.2.2.1- 1.3.6.1.4.1.6347.2.52.1.2.2.20
SNMP_AV_x, AV, y	1.3.6.1.4.1.6347.2.52.1.1.x.1- 1.3.6.1.4.1.6347.2.52.1.1.x.(y-1)	1.3.6.1.4.1.6347.2.52.1.2.x.1- 1.3.6.1.4.1.6347.2.52.1.2.x.(y-1)
SNMP_BV_1, BV, 21	1.3.6.1.4.1.6347.2.53.1.1.1.1- 1.3.6.1.4.1.6347.2.53.1.1.1.20	1.3.6.1.4.1.6347.2.53.1.2.1.1- 1.3.6.1.4.1.6347.2.53.1.2.1.20
SNMP_BV_2, BV, 21	1.3.6.1.4.1.6347.2.53.1.1.2.1- 1.3.6.1.4.1.6347.2.53.1.1.2.20	1.3.6.1.4.1.6347.2.53.1.2.2.1- 1.3.6.1.4.1.6347.2.53.1.2.2.20-
SNMP_BV_x, BV, y	1.3.6.1.4.1.6347.2.53.1.1.x.1- 1.3.6.1.4.1.6347.2.53.1.1.x.(y-1)	1.3.6.1.4.1.6347.2.53.1.2.x.1- 1.3.6.1.4.1.6347.2.53.1.2.x.(y-1)

4.1.2   Custom Configuration

There is no special Data Array naming convention or format instructions.

```
Data_Arrays
Data_Array_Name      , Data_Format   , Data_Array_Length
G_ALM_STAT_1         , Uint16        , 7
G_ALM_CNTR_1         , Uint16        , 6
BRN_CUR_ALM_1        , Uint16        , 253
LOCATION_1           , Byte          , 128
AUX_CNT64_           , Uint32        , 2
DA_CONFIG            , Bit           , 1
```

5 Client Side Configuration

For detailed information on FieldServer configuration, 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 SNMP Server such as a SNMP Agent application.

The configuration file tells the FieldServer about its interfaces, and the routing of data required. In order to enable the FieldServer for SNMP 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: In the tables below, * indicates an optional parameter, with the bold legal value as default.

5.1 Client Side Connection Parameters

Section Title	
Connections	

Column Title	Function	Legal Values
Adapter	Adapter Name.	N1, N2, WLAN[2]
Protocol	Specify protocol used.	SNMP-STD
Public_Community* (Also known as SNMP_Community)	Specify the Server's public community name if the FieldServer is supposed to be in a public domain. It will be used in polls (Get/Set) if there is no Private_Community specified. This is a case sensitive parameter. Any poll with a different community could be discarded by the Server.	Any string up to 255 characters, Public
Private_Community*	Specify the Server's private community name if it is different than Public_Community. The FieldServer will use this community in polls (Get/Set) requests.	Any string up to 255 characters. **If undefined:** value associated with Public_Community.
Trap_Community*	Specify the Server's trap community name if it is different than Public_Community. The FieldServer will discard any trap/inform received with an unknown community. It is case sensitive.	Any string up to 255 characters. **If undefined:** value associated with Public_Community.
Application	This parameter should always be set to "custom".	Custom, **-**
SNMP_Protocol_ Version*	Specify the SNMP protocol version supported by the Server.	V2c, **V1**
SNMP_Trap_Port*	Specify the trap port number if the remote server sends a trap/inform to a UDP port other than the standard (162) port.	1-65534, **162**

Example

```
//   Client Side Connections
Connections
Adapter    , Protocol       , Application    , SNMP_Protocol_Version
N1         , SNMP-STD    , custom         , v2c
```

---

[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.

5.2   Client Side Node Parameters

Section Title		
Nodes		

Column Title	Function	Legal Values
Node_Name	Provide name for Node.	Up to 32 alphanumeric characters
Adapter	Adapter Name.	N1, N2, WLAN[3]
Protocol	Specify protocol used.	SNMP-STD
IP_Address	Specify the Server Node's IP Address (SNMP Agent).	IP Address in decimal format
SNMP_Node_Version*	Specify the SNMP protocol version if the remote server supports a different version than specified on the connection.	V2c, **V1**
SNMP_Node_Trap_Port*	Specify the trap port number if the remote server sends a trap/inform to a UDP port other than the standard (162) port and if different from the SNMP_Trap_Port specified on the connection.	1-65534, **162**
Remote_Node_IP_Port*	Specify the poll/response port number if the remote server listens on a UDP port other than the standard (161) port.	1-65534, **161**

Example

```
//   Client Side Nodes
Nodes
Node_Name    , Node_ID    , Protocol    , Port
Panel1          , 1                , SBT-FSI    , P1
```

5.3   Client Side Map Descriptor Parameters

5.3.1   FieldServer Specific 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 **Section 4**
Data_Array_Offset	Starting location in Data Array.	0 to (Data_Array_Length -1) as specified in **Section 4**
Function	Function of Map Descriptor.	Server, Rdbc, Wrbc, Wrbx

---

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

5.3.2   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 **Section 5.2**
Length	Length of Map Descriptor.	1 – (Data Array length – Data Array Offset)
Parent_Map_Descriptor*	Specify the name of the parent Map Descriptor that holds the previous OID integer.	Name of the previously created Map Descriptor, **-**
SNMP_OID*	Specify the Object Identifier. It could be the child number to the parent or the full OID.	0 to any Integer or dot separated full OID, **-**
Data_Type	Specify the data type of the point in the MIB file.	OCTET STRING, INTEGER, Integer32, Counter, Counter32, Counter64, Gauge, Gauge32, Unsigned32, BITS, INDEX, INDEX_INTEGER, Table, TableEntry, NumberString[4], Trap Store, Specific Trap Type
SNMP_Read_Method*	Specify the command that should be used to read data from Server.	**Get**, Walk, BulkWalk
Trap_Value_Index	Specify which value should be stored from trap message values. Index 0 means first value, 1 means second value, etc.	**0** - 255

---

[4] Driver will convert a string to numbers before storing values, and convert numbers to a string before sending. As there is no basic Float format in SNMP, NumberString can be used to transfer Floating point numbers.

5.4   Map Descriptor Examples

5.4.1   Simple Get/Set Map Descriptors Example

The following Map Descriptor will read the object identified by full OID 1.3.6.1.4.6347.1.1.3 every second and will store the data at offset 2 in DA_ScalarVar Data Array.

If the same Data Array location will be updated by another protocol, the FieldServer will issue a "Set" command to the SNMP server.

```
//   Client Side Map Descriptors

Map_Descriptors
Map_Descriptor_Name        , Data_Array_Name   , Data_Array_Offset     , Function       , Node_Name
Readn1ScalarObj3Name       , DA_SCALARVAR      , 2                     , rdbc           , SNMP_Server
```

```
 , Length       , SNMP_OID                 , Data_Type       , Scan_Interval
 , 1            , 1.3.6.1.4.1.6347.1.1.3   , Integer32       , 1.000s
```

5.4.2   Walk or BulkWalk Map Descriptors Example

Using "Walk" or "BulkWalk" FieldServer can read all the objects under a specified OID.

Data returned by each poll will be stored on passive Map Descriptors. Passive Map Descriptors should be created to build an OID tree structure from the Server's MIB file using the instructions under **Section 6.4.3**, **Section 6.4.4**, **Section 6.4.5** and **Section 6.4.6**.

```
//   Client Side Map Descriptors

Map_Descriptors
Map_Descriptor_Name        , Data_Array_Name   , Data_Array_Offset     , Function       , Node_Name
ReadEverything             , DA_POLLER         , 0                     , rdbc           , SNMP_Server
```

```
 , Length       , SNMP_OID      , Scan_Interval       , SNMP_Read_Method
 , 50           , All           , 5.000s              , BulkWalk
```

In the above example:

- **Length** – If it is a BulkWalk request, the driver will request the next 50 objects to be sent by the Server. If it is a simple walk request, this parameter will get ignored and the Server will only send 1 next object.

- **SNMP_OID** – OID to start with. "All" means the root of the OID tree i.e. 1.3. It could also be any OID, the driver will read the whole tree under this OID.

5.4.3   Storing V1 Customer Map Descriptors Traps Example

V1 traps are mostly vendor specific and each field in the trap message may have a vendor specific meaning. It is possible to store raw values from trap message and it can be transferred or used for custom logic.

The following set of Map Descriptors store the 'specific trap type' from the trap header and the first 3 variable bindings values from the trap message.

```
//   Client Side Map Descriptors

Map_Descriptors

Map_Descriptor_Name      , Data_Array_Name       , Data_Array_Offset      , Function       , Node_Name
v1trap_storage           , DUMMY                 , 0                      , Passive        , SNMP_Server
trap_specific_type       , DA_Specific_Type      , 0                      , Passive        , SNMP_Server
OID1_VALUE               , DA_OID1_VALUE         , 0                      , Passive        , SNMP_Server
OID2_VALUE               , DA_OID2_VALUE         , 0                      , Passive        , SNMP_Server
OID3_VALUE               , DA_OID3_VALUE         , 0                      , Passive        , SNMP_Server
```

```
, Length      , Data_Type                  , Parent_Map_Descriptor      , Trap_Value_Index
, 1           , Trap Store                 , -                          , -
, 1           , Specific Trap Type         , v1trap_storage             , -
, 1           , INTEGER                    , v1trap_storage             , 0
, 22          , OBJECT IDENTIFIER          , v1trap_storage             , 1
, 20          , DisplayString              , v1trap_storage             , 2
```

5.4.4   Receiving Traps from an Agent as an SNMP Client

When configuring the FieldServer as an SNMP client, traps sent by an Agent to the Client will automatically update the related OID as long as the OID has been configured as a map descriptor somewhere in the FieldServer configuration. Consequently, it is not necessary to configure separate map descriptors for traps unless the OID's in question are not already configured. Should a GET request need to be configured to manage a trap that does not already have an associated OID in the configuration, then it is best to configure this trap using a passive map descriptor to receive the trap only (active GET requests may cause errors on Trap OID's if the Agent does not support it).

NOTE: The driver can support traps with full OIDs only[5].

---

[5] Some agents could send traps with relational OID/value pairs, for example: the value of an OID in a trap should be combined to other OID's in the trap (or a standard OID) to make an absolute OID. If the driver does not support it, the driver assumes each OID in an OID/value pair as an absolute OID.

6   Server Side Configuration

For detailed information on FieldServer configuration, 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" files provided with the FieldServer).

This section documents and describes the parameters necessary for configuring the FieldServer to communicate with a SNMP Client such as a Network Management application. Refer to **Section 7.1** for information on how to configure SNMP TRAPS.

The configuration file tells the FieldServer about its interfaces, and the routing of data required. In order to enable the FieldServer for SNMP communications, the driver independent FieldServer buffers need to be declared in the "Data Arrays" section, the FieldServer virtual node(s) needs to be declared in the "Server Side Nodes" section, and the data to be provided to the Clients needs to be mapped in the "Server Side Map Descriptors" section. Details on how to do this can be found below.

NOTE: In the tables below, * indicates an optional parameter, with the bold legal value as default.

6.1   Server Side Connection Parameters

Section Title		
Connections		

Column Title	Function	Legal Values
Adapter	Adapter Name.	N1, N2, WLAN[6]
Protocol	Specify protocol used.	SNMP-STD
Public_Community* (Also known as SNMP_Community)	Specify the public community name. Any device accessing the FieldServer with this community name will have read-only privileges. This is a case sensitive parameter. If community in poll is not the same as public or private, poll will be discarded.	Any string up to 255 characters, Public
Private_Community*	Specify the private community name if it differs from the public_community. Any device accessing the FieldServer with this community name will have read-write privileges. This is a case sensitive parameter.	Any string up to 255 characters. **If undefined:** value associated with Public_Community.
Trap_Community*	Specify the trap community name if it differs from the public_community. This community name will be used in traps/informs generated by FieldServer. This is a case sensitive parameter.	Any string up to 255 characters. **If undefined:** value associated with Public_Community.
SNMP_Trap_Port*	Specify the UDP port number for the Server to send a trap/inform.	1-65534, **162**
IP_Port*	Specify the UDP port number for the Server to accept polls on.	1-65534, **161**

Example

```
//   Server Side Connections
Connections
Adapter    , Protocol       , Application   , SNMP_Protocol_Version
N1         , SNMP-STD    , custom         , v2c
```

---

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

6.2   Server Side Node Parameters

NOTE: Required only if sending Traps or for custom configuration.

Section Title
Remote_Client_Node_Descriptors

Column Title	Function	Legal Values
Node_Name	Provide name for node.	Up to 32 alphanumeric characters
Adapter	Adapter Name.	N1, N2, WLAN[7]
Protocol	Specify protocol used.	SNMP-STD
IP_Address	Specify the Client Node's IP Address (Receiver of Traps).	IP Address in decimal format.

Virtual_IP	If Virtual_IP is full IP Address (e.g.192.168.1.30), it will be added as it is. Otherwise it will be considered as an offset to base adapter IP for first time, and auto increment thereafter. For example, suppose Virtual_IP is 10 and unit's IP is 192.168.1.101:
	    The first virtual IP will be 192.168.1.111
	    Second virtual IP will be 192.168.1.112
	Virtual_IP value 0 or - will not create any virtual IP.
	NOTE: All Virtual IPs should be on the same sub network as of base adapter network. If base adapter IP is 192.168.2.203, netmask is 255.255.255.0 and Virtual_IP is 192.168.1.29. Then netmask should be changed (say 255.255.0.0) to put both IPs on same sub network.
		-, Any IP Address e.g. 192.168.1.45 offset to base IP e.g. 1, 2, 3 ...
		
		NOTE: 0 or - will not create any virtual IP.

Create_MIB*	Specify if MIB file should be created from this node. The MIB file uses the following naming convention: f<ip>_<nn>.mib where <ip> is node's host IP Address (in decimal dot notation format) and <nn> is node name. Refer to **Section 7.4** for instructions on how to upload a MIB file from the FieldServer.	**No**, Yes

Example

```
//   Server Side Nodes

Remote_Client_Node_Descriptors
Node_Name          , Node_ID       , Protocol          , Adapter       , IP_Address          , Create_MIB
SNMP_Server        , 1             , SNMP-STD          , N1            , 192.168.1.17        , Yes
```

---

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

6.3   Server Side Standard Map Descriptor Parameters

NOTE: Required only if sending Traps.

Server Side Map Descriptors are not required by SNMP for Get, Get_Next or Set requests, since the mapping of FieldServer Data Arrays into the SNMP OID addressing scheme follows the method outlined in **Section 4**. Server Side Map Descriptors are required to configure SNMP TRAPS only.

6.3.1   FieldServer Specific 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 **Section 4**
Data_Array_Offset	Starting location in Data Array.	0 to maximum as specified in **Section 4**
Function	Function of Client Map Descriptor.	SNMP_TRAP

6.3.2   Driver Related Map Descriptor Parameters

Column Title	Function	Legal Values
Node_Name	Name of Node or panel for which this Map Descriptor holds points.	One of the Node names specified in **Section 6.2**
Length	Length of Map Descriptor.	1 – (Data Array length – Data Array Offset)
Trap_Type*	Specify the trap type that will be encoded in the Trap message whenever point is in Off-Normal state. Trap type Normal will be encoded in the trap message while the point is in normal state.	Trap_Type Encode value **Alarm** 6 High Alarm 7 Low Alarm 8 Normal 9 Fault 10 Trouble 10 Error 11 Warning 12 Cov 13
Cov_Normal*	This is applicable to Digital points only. Specify the normal value for digital point. When the value changes to or from Normal, a Trap will be sent.	**0**, 1
Cov_Hi_Alm*	Specify High Alarm threshold.	**0.0**, Any float/integer value
Cov_Hi_Warn*	Specify High Warning threshold.	**0.0**, Any float/integer value
Cov_Lo_Warn*	Specify Low Warning threshold.	**0.0**, Any float/integer value
Cov_Lo_Alm*	Specify Low Alarm threshold.	**0.0**, Any float/integer value
Cov_Deadband*	Specify the deadband value which must be exceeded for Alarm or Warning states to toggle (prevents chatter). A Trap will be sent only if the value change exceeds the deadband or the state changes.	**0.0**, Any float/integer value

6.3.3   Binary Trap Map Descriptor Example

The following Map Descriptors will generate a Trap whenever value/state changes from/to normal.

```
//   Server Side Map Descriptors

Map_Descriptors
Map_Descriptor_Name      , Data_Array_Name    , Data_Array_Offset     , Function          , Trap_Type
Trap_Range_BI            , SNMP_BI            , 3                     , SNMP_Trap         , Alarm
Trap_Range_BIb           , SNMP_BI            , 5                     , SNMP_Trap         , Alarm
```

```
, Node_Name       , Length     , Cov_Normal
, Mngr_1          , 2          , 0
, Mngr_1          , 1          , 1
```

Trap of type Alarm (6) will be generated.

A Trap of type Alarm (6) will be generated when the value changes from the specified COV_Normal value, and a Trap of Type Normal (9) will be generated when the value returns to the specified value.

6.3.4   Analog Trap Map Descriptor Example

The following Map Descriptor will generate a Trap whenever a value changes by the deadband (5) or the point's status changes (Normal, Warning, Alarm).

```
//   Server Side Map Descriptors

Map_Descriptors
Map_Descriptor_Name      , Data_Array_Name    , Data_Array_Offset     , Function          , Trap_Type
Trap_Range_AI2           , SNMP_AI            , 2                     , SNMP_Trap         , Alarm
```

```
, Node_Name       , Length     , Cov_Deadband      , Cov_Hi_Warn     , Cov_Hi_Alm
, Mngr_1          , 1          , 5                 , 50             , 100
```

6.3.5   Emulating an MXL/XLS Device with Analogs

The following list of Map Descriptors emulates an MXL Server:

```
// Server Side Map Descriptors

Map Descriptor
Map_Descriptor_Name , Data_Array_Name , Data_Array_Offset , Function , Node_Name , Block_Number , Length , Msg_Type
SD_Analog_Volts1     , Volts           , 0                 , Passive  , Panel1     , 1            , 252    , Analog_Volts
SD_Analog_Sense1     , Sense           , 0                 , Passive  , Panel1     , 1            , 252    , Analog_Sense
SD_Analog_Thres1     , Thres           , 0                 , Passive  , Panel1     , 1            , 252    , Analog_Thres
```

In the above example:

- **Data_Array_Name** – Use Float Data Arrays for the analog Map Descriptors to get the full analog value.

6.4 Custom Server Side Map Descriptors

6.4.1 FieldServer Related Map Descriptor Parameters

Column Title	Function	Legal Values
Map_Descriptor_Name	Name of this Map Descriptor. Name of this Map Descriptor is used as the object name for the MIB file.	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 **Section 4**
Data_Array_Offset	Starting location in Data Array.	0 to maximum as specified in **Section 4**
Function	Function of Client Map Descriptor.	SNMP_TRAP

6.4.2 Driver Specific Map Descriptor Parameters

Section Title		
Map Descriptors		

Column Title	Function	Legal Values
Node_Name	Name of the Remote Client Node to which Trap will be sent.	One of the Node names specified in **Section 6.2**
Length	Length of Map Descriptor.	1 – (Data Array length – Data Array Offset)
SNMP_OID	Specify the Object Identifier. One Map Descriptor is required for each OID specified as an integer.	0 to any Integer
Access_Level	Specify if the OID is read only or read/write. If not specified, it defaults to read only.	**Read-only**, Read-write, -
Parent_Map_ Descriptor*	Specify the name of the parent Map Descriptor that holds the previous OID integer.	Name of the previously created Map Descriptor, **-**
Data_Type	Specify the data type of the point in the MIB file.	OCTET STRING, INTEGER, Integer32, Counter, Counter32, Counter64, Gauge, Gauge32, Unsigned32, BITS, INDEX, INDEX_INTEGER, Table, TableEntry, Trap, TrapVar, NumberString[8]
SNMP_ TRAP_MD	Specify the name of the TrapVar Map Descriptor to which this trap Map Descriptor is linked.	One of the Map Descriptors with data type set to TrapVar configured previously
Inform_Request*	Specify if this is an Inform Request or not. Inform Requests are traps required to be acknowledged.	Yes[9], **No**
Description*	Specify the description about the Map Descriptor. This is used as the 'Description' clause in the MIB file.	Text up to 199 Characters. Default is **""**

---

[8] Driver will convert a string to numbers before storing values and convert numbers to a string before sending. As there is no basic Float format in SNMP, NumberString can be used to transfer Floating point numbers.
[9] When set to yes, a MIB file is auto generated and stored in the FieldServer.

6.4.3 Basic OID Structure Map Descriptors Example

For the following MIB file structure the support for iso.org.dod.internet.private.(1.3.6.1.) OID is built-in:

```
verisIndustries  OBJECT IDENTIFIER ::= { enterprises 40845 }  --  1.3.6.1.4.1.40845 Veris Private Enterprise
Number
  verisEnergy      OBJECT IDENTIFIER ::= { verisIndustries 1 }  --  1.3.6.1.4.1.40845.1
    bcpmE30        OBJECT IDENTIFIER ::= { verisEnergy 30 }  --  1.3.6.1.4.1.40845.1.30
      panels       OBJECT IDENTIFIER ::= { bcpmE30 1 }  --  1.3.6.1.4.1.40845.1.30.1
        panel1       OBJECT IDENTIFIER ::= { panels 1 }  --  1.3.6.1.4.1.40845.1.30.1.1
          p1Alarms       OBJECT IDENTIFIER ::= { panel1 1 }  --  1.3.6.1.4.1.40845.1.30.1.1.1
                    p1VoltageInputs   OBJECT IDENTIFIER ::= { panel1 2 }  --
                    1.3.6.1.4.1.40845.1.30.1.1.2
          p1AuxiliaryInputs OBJECT IDENTIFIER ::= { panel1 3 }  --  1.3.6.1.4.1.40845.1.30.1.1.3
```

The customized configuration is shown below:

```
//   Server Side Map Descriptors


Map_Descriptors
```

Map_Descriptor_Name	, Data_Array_Name	, Data_Array_Offset	, Function	, Node_Name
private	, DA_CONFIG	, 0	, Server	, Agent1
enterprise	, DA_CONFIG	, 0	, Server	, Agent1
verisIndustries	, DA_CONFIG	, 0	, Server	, Agent1
verisEnergy	, DA_CONFIG	, 0	, Server	, Agent1
bcpmE30	, DA_CONFIG	, 0	, Server	, Agent1
Panels	, DA_CONFIG	, 0	, Server	, Agent1
Panel1	, DA_CONFIG	, 0	, Server	, Agent1
p1Alarms	, DA_CONFIG	, 0	, Server	, Agent1
p1VoltageInputs	, DA_CONFIG	, 0	, Server	, Agent1
P1AuxilaryInputs	, DA_CONFIG	, 0	, Server	, Agent1

, Length	, SNMP_OID	, Parent_Map_Descriptor	, Data_Type
, 1	, 4	, -	, -
, 1	, 1	, private	, -
, 1	, 40845	, enterprise	, -
, 1	, 1	, verisIndustries	, -
, 1	, 30	, verisEnergy	, -
, 1	, 1	, bcpmE30	, -
, 1	, 1	, panels	, -
, 1	, 1	, panel1	, -
, 1	, 2	, panel1	, -
, 1	, 3	, panel1	, -

6.4.4 Simple Object Map Descriptor Example

The following section of an MIB file shows simple child (i.e. last in tree) OIDs of p1Alarms:

```
p1GlobalAlarmStatus OBJECT IDENTIFIER ::= { p1Alarms 1 }
   p1TotalLatchingChannelsInAlarm OBJECT-TYPE
      SYNTAX          Unsigned32
      MAX-ACCESS        read-only
      STATUS          current
      DESCRIPTION       "Total Number of Aux and Branch Channels in Latching Alarm"
      ::= { p1GlobalAlarmStatus 5 }

   p1TotalNonLatchingChannelsInAlarm OBJECT-TYPE
      SYNTAX          Unsigned32
      MAX-ACCESS        read-only
      STATUS          current
      DESCRIPTION       "Total Number of Aux and Branch Channels in Non-Latching Alarm"
      ::= { p1GlobalAlarmStatus 6 }
```

The following three Map Descriptors must be defined for this section of the MIB file:

```
//   Server Side Map Descriptors

Map_Descriptors
Map_Descriptor_Name                  , Data_Array_Name      , Data_Array_Offset      , Function
p1GlobalAlarmStatus                  , DA_CONFIG            , 0                      , Server
p1TotalLatchingChannelsInAlarm       , G_ALM_STAT_1         , 4                      , Server
P1TotalNonLatchingChannelsInAlarm    , G_ALM_STAT_1         , 5                      , Server
```

```
, Node_Name      , Length    , SNMP_OID      , Parent_Map_Descriptor              , Data_Type
, Agent1         , 1         , 1             , p1Alarms                           , -
, Agent1         , 1         , 5             ,p1GlobalAlarmStatus                 , Unsigned32
, Agent1         , 1         , 6             , p1TotalLatchingChannelsInAlarm     , Unsinged32
```

6.4.5 Table, TableEntry and Objects in TableEntry Map Descriptor Example

The following section for an MIB file shows a table of p1Alarms:

```
p1AuxVoltageAndCurrentAlarmTable OBJECT-TYPE
     SYNTAX          SEQUENCE OF P1AuxVoltageAndCurrentAlarmTableEntry
     MAX-ACCESS       not-accessible
     STATUS          current
     DESCRIPTION      "Individual Auxiliary Phase Alarms: Index 1, 2, 3, 4 = Phase A, B, C, N"
     ::= { p1Alarms 3 }
```

The following Map Descriptor implements the table:

```
//   Server Side Map Descriptors

Map_Descriptors
Map_Descriptor_Name                    , Data_Array_Name   , Data_Array_Offset   , Function
p1AuxVoltageAndCurrentAlarmTable      , DA_CONFIG          , 0                    , Server
```

```
, Node_Name       , Length     , SNMP_OID      , Parent_Map_Descriptor     , Data_Type
, Agent1          , 1          , 3             , p1Alarms                  , Table
```

The following section in the MIB file shows a table entry:

```
p1AuxVoltageAndCurrentAlarmTableEntry  OBJECT-TYPE
     SYNTAX          P1AuxVoltageAndCurrentAlarmTableEntry
     MAX-ACCESS       not-accessible
     STATUS          current
     DESCRIPTION      "Individual Auxiliary Phase Alarms: Index 1, 2, 3, 4 = Phase A, B, C, N"
     INDEX          {
                   p1AuxAlarmIndex
               }
     ::= { p1AuxVoltageAndCurrentAlarmTable 1 }
```

The following Map Descriptor implements the table entry:

```
//   Server Side Map Descriptors

Map_Descriptors
Map_Descriptor_Name                         , Data_Array_Name   , Data_Array_Offset , Function    , Node_Name
p1AuxVoltageAndCurrentAlarmTableEntry       , DA_CONFIG          , 0                  , Server      , Agent1
```

```
, Length      , SNMP_OID      , Parent_Map_Descriptor                , Data_Type
, 1           , 1             , p1AuxVoltageAndCurrentAlarmTable      , TableEntry
```

The following section in the MIB file shows a table entry that doesn't need a defined Map Descriptor:

```
P1AuxVoltageAndCurrentAlarmTableEntry ::= SEQUENCE {
        p1AuxAlarmIndex  -- index 1, 2, 3, 4 = phase A, B, C, N
           Unsigned32,
        p1AuxAlarmPhase  -- index 1, 2, 3, 4 = phase A, B, C, N
           Unsigned32,
        p1VoltagePhaseAlarmStatus -- [241-243]   voltage alarms
           BITS,
        p1AuxCurrentAlarmStatus -- [220-223]  current alarms
           BITS,
        p1AuxHighHighAlarmCount -- [343-346]
           Counter32,
        p1AuxHighAlarmCount -- [389-392]
           Counter32,
        p1AuxLowAlarmCount -- [435-438]
           Counter32,
        p1AuxLowLowAlarmCount -- [481-484]
           Counter32,
        p1AuxOffStateAlarmCount -- [527-530]
           Counter32
```

For the following example each entry in tableEntry has been defined as an individual object and needs a Map Descriptor for each object:

```
     p1AuxAlarmIndex OBJECT-TYPE    --  index 1, 2, 3, 4 = phase A, B, C, N
SYNTAX          Unsigned32 (1..4)
MAX-ACCESS       not-accessible
STATUS          current
DESCRIPTION       "Auxiliary Phase Number Index: 1, 2, 3, 4 = phase A, B, C, N"
::= { p1AuxVoltageAndCurrentAlarmTableEntry 1 }

   p1AuxAlarmPhase OBJECT-TYPE    --  index 1, 2, 3, 4 = phase A, B, C, N (same as
Index, but readable)
SYNTAX          Unsigned32 (1..4)
MAX-ACCESS       read-only
STATUS          current
DESCRIPTION       "Auxiliary Phase Number Index: 1, 2, 3, 4 = phase A, B, C, N"
::= { p1AuxVoltageAndCurrentAlarmTableEntry 2 }
```

The following two Map Descriptors are defined by index in the table but have different names and might not be connected to any real time data. In some MIB files there could be only single index entry. Correspondingly Map descriptors will be as follows:

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

Map_Descriptors				
Map_Descriptor_Name	, Data_Array_Name	, Data_Array_Offset	, Function	, Node_Name
p1AuxAlarmIndex	, DA_CONFIG	, 0	, Server	, Agent1
p1AuxAlarmPhase	, DA_CONFIG	, 0	, Server	, Agent1

, Length	, SNMP_OID	, Parent_Map_Descriptor	, Data_Type
, 4	, 1	, p1AuxVoltageAndCurrentAlarmTableEntry	, INDEX
, 4	, 2	, p1AuxVoltageAndCurrentAlarmTableEntry	, INDEX

Other objects defined in a table entry might be simple objects.

Looking at the last two objects from tableEntry as an example:

```
p1AuxLowLowAlarmCount OBJECT-TYPE    --  [481-484]
        SYNTAX          Counter32
        MAX-ACCESS       read-only
        STATUS          current
        DESCRIPTION      "Count of Auxiliary Current Low Low Alarms"
        ::= { p1AuxVoltageAndCurrentAlarmTableEntry 8 }

    p1AuxOffStateAlarmCount OBJECT-TYPE    --  [527-530]
        SYNTAX          Counter32
        MAX-ACCESS       read-only
        STATUS          current
        DESCRIPTION      "Count of Auxiliary Current Alarm Off State Transitions"
        ::= { p1AuxVoltageAndCurrentAlarmTableEntry 9 }
```

Corresponding Map Descriptors will be as follows:

```
//   Server Side Map Descriptors

Map_Descriptors
```

Map_Descriptor_Name	, Data_Array_Name	, Data_Array_Offset	, Function	, Node_Name
p1AuxLowLowAlarmCount	, AUX_VC_ALM_1	, 21	, Server	, Agent1
p1AuxOffStateAlarmCount	, AUX_VC_ALM_1	, 25	, Server	, Agent1

, Length	, SNMP_OID	, Parent_Map_Descriptor	, Data_Type
, 4	, 8	, p1AuxVoltageAndCurrentAlarmTableEntry	, Counter32
, 4	, 9	, p1AuxVoltageAndCurrentAlarmTableEntry	, Counter32

6.4.6   String Object Map Descriptor Example

The MIB file below shows a string object entry:

```
    p1LocationString   OBJECT-TYPE    --  [7-70]
    SYNTAX          DisplayString (SIZE(128))
    MAX-ACCESS       read-write
    STATUS          current
    DESCRIPTION      "Null terminated Text String Describing Physical Installation"
    ::= { p1ProductInformation 5 }
```

The corresponding Map Descriptor will be as follows. Note parent map descriptor p1ProductInformation should be defined before this Map Descriptor.

Although the Map Descriptor length is 128, the driver will only serve characters until the null character in the LOCATION_1 Data Array. The null character will not be included in the response string.

```
//   Server Side Map Descriptors

Map_Descriptors
```

Map_Descriptor_Name	, Data_Array_Name	, Data_Array_Offset	, Function	, Node_Name
p1LocationString	, LOCATION_1	, 0	, Server	, Agent1

, Length	, SNMP_OID	, Parent_Map_Descriptor	, Data_Type
, 128	, 5	, p1ProductInformation	, DisplayString

6.4.7   Trap/Notification Map Descriptor Example

The MIB file below shows the definition of a trap:

```
p1Traps OBJECT IDENTIFIER ::= { p1Alarms 0 }
```

The corresponding Map Descriptor will be:

// Server Side Map Descriptors				
Map_Descriptors				
Map_Descriptor_Name	, Data_Array_Name	, Data_Array_Offset	, Function	, Node_Name
p1Traps	, DA_CONFIG	, 0	, Server	, Agent1

, Length	, SNMP_OID	, Parent_Map_Descriptor	, Data_Type
, 1	, 0	, p1Alarms	,TRAP

A next entry in the MIB file declares the trap to include p1AuxAlarmPhase and p1VoltagePhaseStatus as trap variables:

```
p1VoltagePhaseAlarmStatusTrap NOTIFICATION-TYPE    -- [241-243]   voltage alarms A, B,
C. Return phase index & status register.
      OBJECTS
      {
         p1AuxAlarmPhase,
         p1VoltagePhaseAlarmStatus
      }
      STATUS          current
      DESCRIPTION        "Voltage Alarm Status Bit Map: 0 HighLatching, 1 LowLatching,
8 HighNonLatching"
      ::= { p1Traps 3 }
```

The corresponding Map Descriptors will be as follows:

// Server Side Map Descriptors				
Map_Descriptors				
Map_Descriptor_Name	, Data_Array_Name	, Data_Array_Offset	, Function	, Node_Name
p1VoltagePhaseAlarmStatusTrap	, DA_CONFIG	, 0	, Server	, Agent1
p1AuxAlarmIndex	, AUX_VC_ALM_1	, 0	, Server	, Agent1
p1VoltagePhaseAlarmStatus	, AUX_VC_ALM_1	, 1	, Server	, Agent1

, Length	, SNMP_OID	, Parent_Map_Descriptor	, Data_Type
, 1	, 3	, p1Traps	, TRAP
, 1	, 1	, VoltagePhaseAlarmStatusTrap	, TrapVar
, 4	, 2	, VoltagePhaseAlarmStatusTrap	, TrapVar

Once the trap and trap variable Map Descriptors have been defined, a Map Descriptor is needed to trigger the trap.

The following Map Descriptor will trigger the trap whenever any value changes in Data Array AUX_VC_ALM_1 offset 1 thru 4.

This Map Descriptor will trigger p1VoltagePhaseAlarmStatusTrap that has been defined above and it will include two variables in the trap.

Map_Descriptors			
Map_Descriptor_Name	, Data_Array_Name	, Data_Array_Offset	, Function
p1VoltagePhaseAlarmStatus	, AUX_VC_ALM_1	, 1	, SNMP_Trap

, Node_Name	, Length	, SNMP_Trap_MD
, Agent1	, 4	, p1VoltagePhaseAlarmStatusTrap

7　Useful Features

7.1　Contents of Trap Messages

Each Trap has a Specific-Trap and 4 OIDs for four bound variables (the payload). Systems that use traps are typically set up to watch for OID's in the payloads. The Specific-Trap is set according to the Map Descriptor's Trap_Type parameter.

The four bound variables are as specified in standard MIB file as:

VARIABLES {notificationPointType, notificationPointIndex, notificationPointDescription, notificationPointValue}

Variable	OID
notificationPointType OBJECT-TYPE 　　　SYNTAX　　　　INTEGER　　(analogInput(1), analogOutput(2),　　　analogValue(3),　binaryInput(4), binaryOutput(5),　binaryValue(6)　} 　　　ACCESS　read-only 　　　STATUS　current 　　　DESCRIPTION　"Identifies　the　Point　Type　in　a Notification." 　　　::= {notificationFields 1 }	1.3.6.1.4.6347.2.1.1.0 ,
notificationPointIndex OBJECT-TYPE 　　　SYNTAX　INTEGER 　　　ACCESS　read-only 　　　STATUS　current 　　　DESCRIPTION　"Identifies　the　Point　Index　in　a Notification." 　　　::= {notificationFields 2 }	1.3.6.1.4.6347.2.1.2.0 ,
notificationPointDescription OBJECT-TYPE 　　　SYNTAX　OCTET STRING 　　　ACCESS　read-only 　　　STATUS　current 　　　DESCRIPTION　"Description　of　the　Point　in　a Notification." 　　　::= {notificationFields 3 }	1.3.6.1.4.6347.2.1.3.0 ,
notificationPointValue OBJECT-TYPE 　　　SYNTAX　INTEGER 　　　ACCESS　read-only 　　　STATUS　current 　　　DESCRIPTION　"Value　of　the　Point　in　a Notification." 　　　::= {notificationFields 4 }	1.3.6.1.4.6347.2.1.4.0 ,

7.1.1 Trap Example

This Trap was sent for Binary Input 3 (named as "BI value 3") when the value changed to 1. The Specific Trap for this Trap is 6 which is "Alarm" The OID's for the 4 bound variables are as follows:

OID	Format	Value	Description
1.3.6.1.4.1.6347.2.1.1.0	Integer	4	Type object (4=Binary Input)
1.3.6.1.4.1.6347.2.1.2.0	Integer	3	Object number
1.3.6.1.4.1.6347.2.1.3.0	OctetString	BI value 3	Object name preloaded in config file
1.3.6.1.4.1.6347.2.1.4.0	Integer	1	Current Object value

The following snapshot shows this Trap in Wireshark capture:
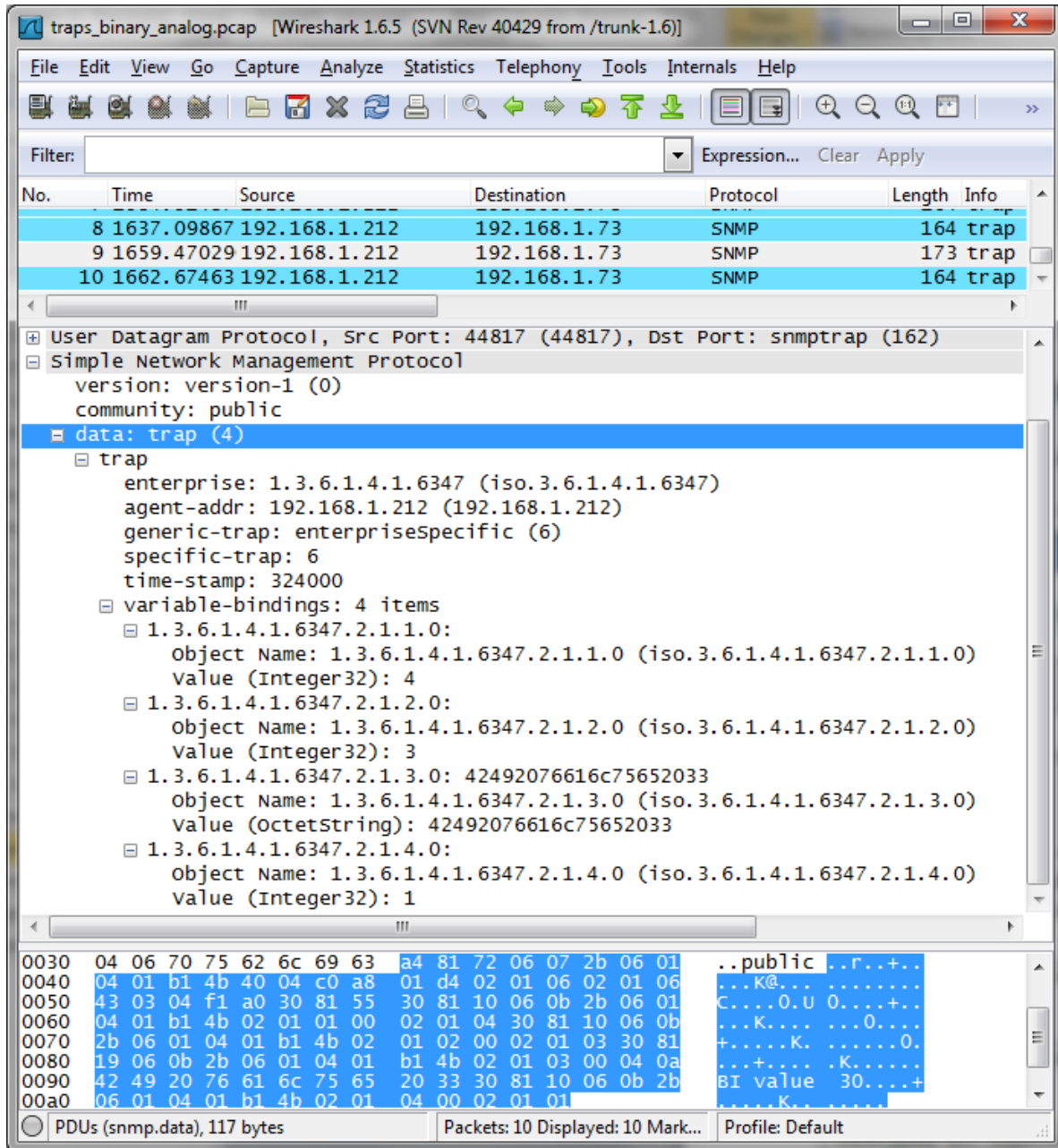
7.2   Using an .ini File to Set the Enterprise_ID and Enterprise_Name

The vendor.ini file can be used to change the SNMP_enterprise_id and SNMP_enterprise_name if desired. A file with the following format must be created and downloaded to the FieldServer:

SNMP_enterprise_id   =   6347
SNMP_enterprise_name = FieldServertechnologies

NOTE: When using vendor.ini, also update FServer.mib file. Replace all instances of FieldServertechnologies with new enterprise name and id 6347 to new id.

7.3   Preloading Point Descriptions

The following example shows how to preload description strings for SNMP points:

```
Preloads
Data_Array_Name            , Preload_Data_Index    , Preload_Data_Format    , Preload_Object_Name
// Analog Input/Output/value description
SNMP_AI                    , 3                      , String                 , AI value 3 Desc
SNMP_AO                    , 4                      , String                 , AO value 4 Desc
SNMP_AV                    , 5                      , String                 , AV value 5 Desc
// Analog vector1 description
SNMP_AV_1                  , 3                      , String                 , AVector1 val3 Desc
SNMP_AV_1                  , 5                      , String                 , AVector1 val5 Desc
```

7.4   Uploading a MIB File from FieldServer

- Determine the name of the MIB file by going to the FieldServer's FS GUI page

- Click User Messages and then on the Driver tab to see the name of the MIB file that was generated.

- Once the name of the MIB file is known, Enter the IP Address followed by '\' and the name of the MIB file into the web browser address bar.

  o For example: 192.168.2.101\f192.168.2.101_SNMP_Server.csv

8   Reference

8.1   Driver Error Messages

Error #	Msg Screen	Screen message	Meaning	Suggested Solution
SNMP-STD#01	ERROR	SNMP-STD#01: Bad Node_ID - forcing to 1	Node_ID is not in range 1-255. Driver defaulting it to 1.	Update configuration file to assign unique Node_ID in range 1-255. Node_ID's are used when Node_Status bits are prepared.
SNMP-STD#02	DRIVER	SNMP-STD#02 : Could not read SNMP response version	The driver found an error in the message structure preventing it from extracting the SNMP version number.	If this message is received rarely and data is being transferred correctly then ignore it. If it is repeated or affects data transfer then take a log and call tech Support. This message is usually preceded by a message which provides more specific information as to why the parse failed.
SNMP-STD#03	DRIVER	SNMP-STD#03 : Version. Expected=1 Rcvd=%d	This driver only supports SNMP version #1.	
SNMP-STD#4A	DRIVER	SNMP-STD#4A : Extract Int Failed. Index=%d	The driver is extracting an integer value from the SNMP message but the field was incorrectly formatted.	
SNMP-STD#4B	DRIVER	SNMP-STD#4B : Expected INT. Found NULL. Index=%d		
SNMP-STD#05	DRIVER	SNMP-STD#05 : Parse Failed. MD=%s	The driver is attempting to process a SNMP message that is not correctly formatted; contains an error or contains unsupported SNMP elements. Immediately after the message has been printed the driver does a hexadecimal dump of the message.	

Error #	Msg Screen	Screen message	Meaning	Suggested Solution
SNMP-STD#06	DRIVER	SNMP-STD#06: Can't send to Agent/Trap Dest. %s at %s	The FieldServer cannot reach the specified remote Node. The message prints the Node name and the IP Address. This message is most commonly printed when an incorrect IP Address has been specified or when the remote Node is off or in an Error State.	Check the specified IP Address or remote device if it is healthy and connected to network.
SNMP-STD#07	DRIVER	SNMP-STD#07 : Invalid SNMP PDU type 0x%x	An unsupported PDU Type was received by the Server Supported PDU Types: GET/SET/GET-NEXT/TRAP	Refer to msg #05 for follow up actions.
SNMP-STD#08	DRIVER	SNMP-STD:#08 : Bad community string	The SNMP message contains a badly formatted community string or the community string is absent.	If this message occurs occasionally, ignore it. Otherwise take a log and call tech support.
SNMP-STD#09	DRIVER	SNMP:#09 ERR. Couldn't respond to Get Request for OID:	The driver can only serve data for configured OID points.	Ignore this message if it occurs once per full walk test. If this message occurs frequently and for any other request either configure FieldServer or remote device.
SNMP-STD#10	DRIVER	SNMP-STD#10 : Store Int. DA=%s:%d value=%d gen=%d SNMP-STD#10: Store ! String DA=%s:%d value=%d	The driver is storing data. The message indicates where the data was stored. It is intended as a debugging tool.	This message is for information only No Action required.
SNMP-STD#11	DRIVER	SNMP-STD#11: Unsupported data_type: 0x%x	The driver is storing data but the data type is not supported.	Check the Driver fact for supported Data Types and correct the configuration of the remote client/agent.

Error #	Msg Screen	Screen message	Meaning	Suggested Solution
SNMP-STD#12	DRIVER	SNMP-STD#12 : No SNMP-STD DA's configured.	SNMP-STD Data Arrays are missing. There won't be any communication between FieldServer and remote Client.	Declare the Data Arrays as described in **Section 4**.
SNMP-STD#13	ERROR	DRV->SNMP-STD#13 : Standard Data Arrays must be of Complex Data Object type. Set Data_Format to AI, AO, AV, BI, BO or BV	The Data format of the SNMP Data Array is incorrect.	Set correct Data_Format as described in **Section 4**.
SNMP-STD#14	DRIVER	SNMP-STD#14: Couldn't parse msg: found %#x at %d	The Server received a message containing an unsupported bound variable, e.g. OPAQUE, TRAP. The driver prints a hex dump of the offending message.	Ensure that the remote SNMP client is correctly configured. Capture a diagnostic log and call technical support.
SNMP-STD#15	DRIVER	SNMP-STD#15: Invalid Msg	The driver has received a message from a SNMP client which is not correctly formatted or which contains bound variables/features not supported by the driver. This particular error message is printed when the SNMP message does not begin with 0x30.	If this message is received rarely and data is being transferred correctly then ignore it. If it is repeated or affects data transfer then take a log and call tech Support. This message is usually preceded by a message which provides more specific information as to why the parse failed.
SNMP-STD#16	DRIVER	SNMP-ST#16: Message length error	The SNMP message indicates that a message length is different to the number of bytes received.	
SNMP-STD#17	DRIVER	SNMP-STD#17: Could not get Request ID	The SNMP message number is badly formatted or absent.	
SNMP-STD#18	DRIVER	SNMP-STD#18: Could not get Error Status	The SNMP error status field is badly formatted or absent.	
SNMP-STD#19	DRIVER	SNMP-STD#19: Could not get Error Index	The SNMP error id field is badly formatted or absent	

Error #	Msg Screen	Screen message	Meaning	Suggested Solution
SNMP-STD#20	DRIVER	SNMP-STD#20: Expected SNMP_STD_TYPE_SEQ	The message header is correctly formatted but as the driver continues parsing the message it can't find the bound variables.	
SNMP-STD#21	ERROR	DRV->SNMP-STD#21 : Out of udp sockets on port=%d	The FieldServer has no UDP socket or cannot bind it to a particular port. It is possible that the FieldServer is running out of memory.	Take a diagnostic log and call technical support.
SNMP-STD#22		DRV->SNMP-STD#22 : Out of udp sockets		
SNMP-STD#23		DRV->SNMP-STD#23 : Socket binding error on port=%d		
SNMP-STD#51	DRIVER	SNMP-STD#51: Not enough space on DA=%s, reqd length=%d	Data array is not long enough.	This message is printed by the client side of the driver. The Client side of the driver is implemented only to test the Server side. Configure the FieldServer as a Server.
SNMP-STD#52	DRIVER	SNMP-STD#52 : Received set/get/getnext (%d) request	The Client side of the driver has received a request. It should only receive responses.	
SNMP-STD#53	DRIVER	SNMP-STD#53: Ignored received request (%d)		
SNMP-STD#54	DRIVER	SNMP-STD#54: Not a valid SNMP response	The Client side of driver has received a message which is not a valid SNMP response.	
SNMP-STD#55	DRIVER	SNMP-STD#55: Encoded msg length (%d) > (%d) bytes received	Partial message is received or length was not correctly formatted in message.	
SNMP-STD#56	DRIVER	SNMP-STD#56: Response report err=%d MD=%s	A response from the remote device reports an error. The message is good but inside the message there is a field reporting an error.	
SNMP-STD#57	DRIVER	SNMP-STD#57: Incorrect variable binding format	The driver has received a message which is incorrectly formatted or contains unsupported bound variables/features. This particular error message is printed when the SNMP message does not begin with 0x30.	

Error #	Msg Screen	Screen message	Meaning	Suggested Solution
SNMP-STD#58	DRIVER	SNMP-STD#58: Can't parse msg: found %#x at %d	A response to a poll from the FieldServer contains a bound variable that is not supported or expected. For example, an IP_ADDRESS and OPAQUE variable or a TRAP. Immediately after this message the driver prints a hex dump of the offending message.	This message is printed by the client side of the driver. The Client side of the driver is implemented only to test the Server side. Configure the FieldServer as a Server.
SNMP-STD#59	DRIVER	SNMP-STD#59 : Could not read SNMP response version	The driver found an error in the message structure preventing it from extracting the SNMP version number.	
SNMP-STD#60	DRIVER	SNMP-STD#60: Bad community string	The SNMP message contains a badly formatted community string or the community string is absent.	
SNMP-STD#61	DRIVER	SNMP-STD#61: Version. Expected=1 Rcvd=%d	This driver only supports SNMP version #1.	