FieldServer

FS-8700-110 Gamewell-FCI 7200 Series

Driver Manual

(Supplement to the FieldServer Instruction Manual)

APPLICABILITY & EFFECTIVITY

Effective for all systems manufactured after March 2017.

Driver Revision: 1.00

Document Revision: 5.A

Technical Support

Please call us for any technical support needs related to the FieldServer product.

Sierra Monitor Corporation
1991 Tarob Court
Milpitas, CA 95035

Website: www.sierramonitor.com

U.S. Support Information:

+1 408 262-6611
+1 800 727-4377

Email: support@sierramonitor.com

EMEA Support Information:

+44 2033 1813 41

Email: support.emea@sierramonitor.com

TABLE OF CONTENTS

1 FCI 7200 SERIES DRIVER DESCRIPTION

The FCI 7200 Series System Control Units (SCU) are manufactured by Fire Control Instruments. A SCU with an enabled serial port can transmit data to a FieldServer which can, in turn, make the data available to other devices including those which communicate using different protocols (e.g. BACnet).

This passive Client driver does not poll for data, nor does it send data or commands to the SCU. Messages received from the SCU are ignored or stored on the FieldServer depending on the status of the panel. The method of message processing and location on the FieldServer is determined in the FieldServer configuration file. Once stored in the FieldServer the data is available to be read or written using other protocols.

No automatic panel data synchronization technique exists. The data in the FieldServer and the panel status have to be synchronized manually. This typically requires a panel reset.

Since the driver cannot send data or commands to the SCU it cannot be used to acknowledge, silence or reset alarms and other events.

The driver can process the single line messages sent from SCU firmware versions earlier than 2.20 and 3 line messages produced in firmware versions 2.20 and later. Processing of 3 line messages requires the 20 character System ID label to be defined.

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 SCU and is thus not fully documented. If Server side functionality needs to be documented and enhanced, contact the Sierra Monitor sales group.

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

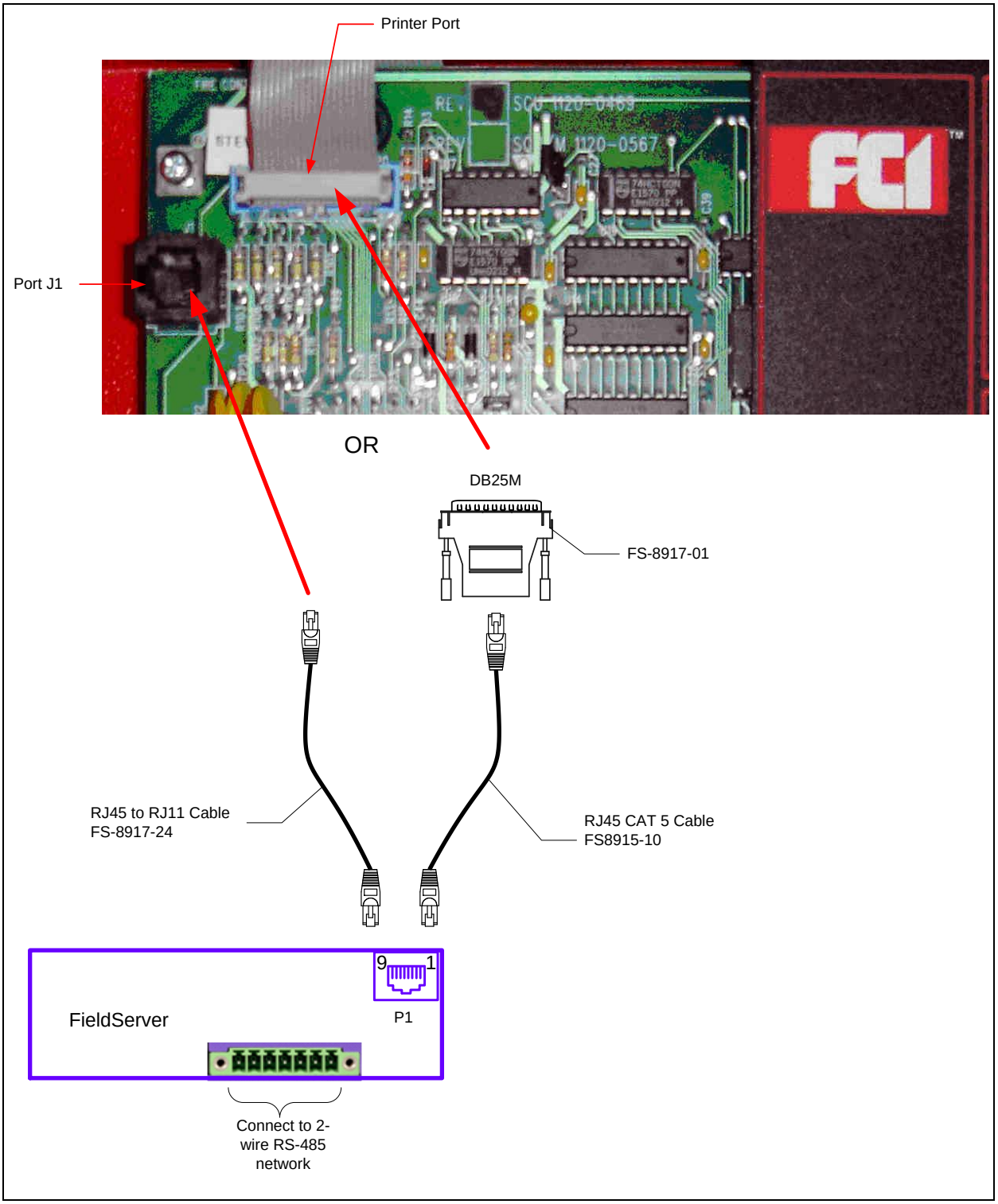
2.1 Supplied by Sierra Monitor

PART #	Description
FS-8915-10	UTP cable (7 foot) for Ethernet connection
FS-8917-01	Conn, DB25M to DCE, RTS/CTS loop
FS-8917-24	RJ45-RJ11/12 Cable assembly for FS connection to FCI panel

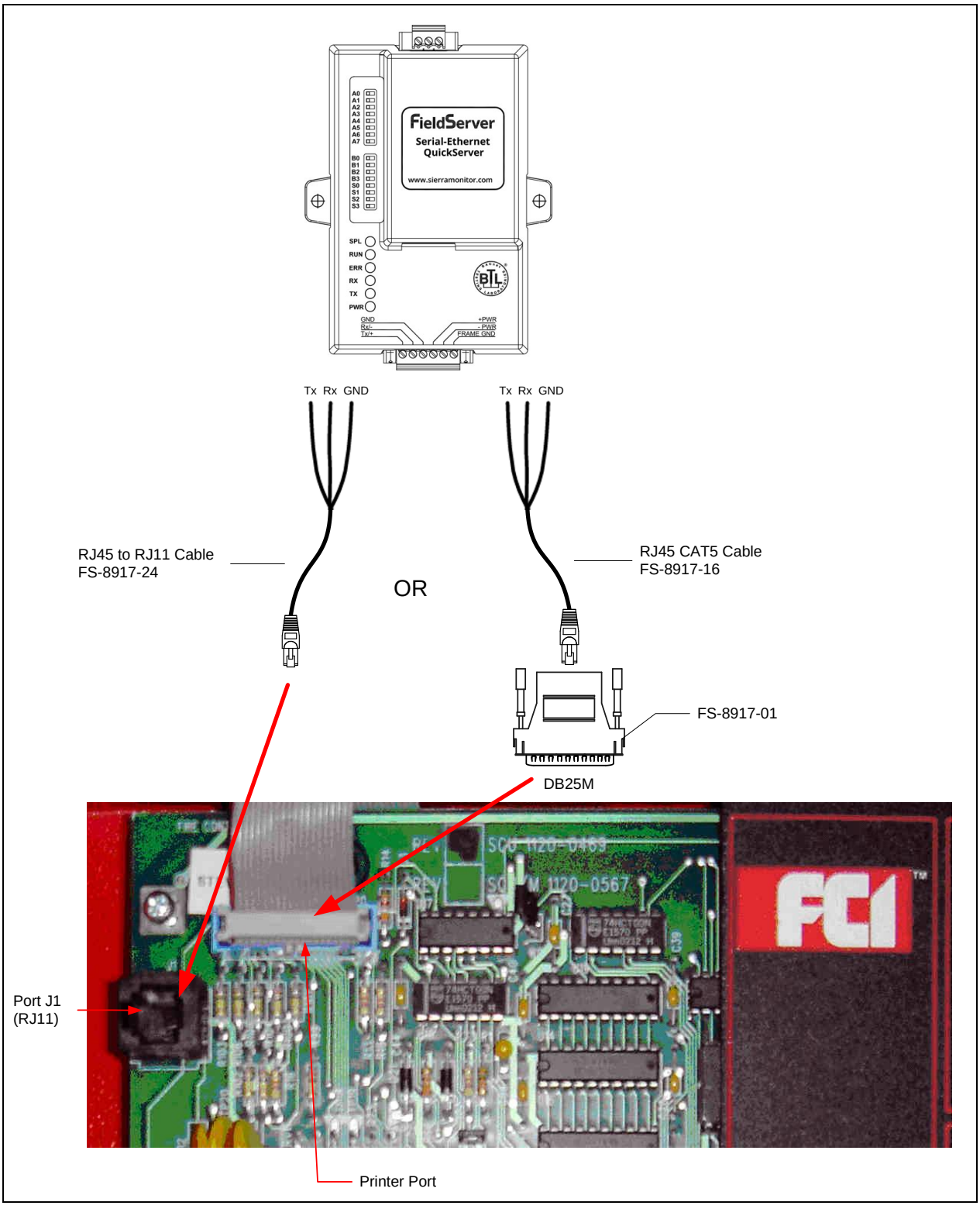
3 HARDWARE CONNECTIONS

The FieldServer is connected to the FCI panel as shown in the connection drawing.

Fieldserver FS-B35:



QuickServer:



3.1 Pinouts

FS-8917-01			
Function	**From**	**To**	**Color**
RX	RJ45-01	DB25M - 03	White
CTS	RJ45-02	DB25M - 05	Brown
DSR	RJ45-03		Yellow
GND	RJ45-04	DB25M - 07	Green
GND	RJ45-05		Red
DSR	RJ45-06		Black
CTS	RJ45-07	DB25M - 04	Orange
TX	RJ45-08	DB25M - 02	Blue

FS-8917-24		
From	**To**	**Color**
RJ11-02	RJ45-01	Black
RJ11-03	RJ45-04	Red
RJ11-05	RJ45-08	Yellow

FS-8917-16		
Signal	**Wire**	**Color**
Rx	RJ45-01	Brown
GND	RJ45-04	Blue/White
Tx	RJ45-08	Orange/White

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, Uint16, Sint16, 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_Array_Format    , Data_Array_Length
DA_AI_01           , UInt16               , 200
DA_AO_01           , UInt16               , 200
DA_DI_01           , Bit                  , 200
DA_DO_01           , Bit                  , 200
```

5 CONFIGURING THE FIELDSERVER AS A FCI 7200 SERIES SERIAL DRIVER CLIENT

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 FCI 7200 Series SCU.

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

5.1 Client Side Connection Descriptions

Section Title		
Connections		
Column Title	**Function**	**Legal Values**
Port	Specify which port the device is connected to the FieldServer.	P1-P2, R1-R2[1]
Protocol	Specify protocol used.	FCI_7200, FCI_7200_series, FCI7200
Baud*	Specify baud rate.	**1200** (Vendor Limitation)
Parity*	Specify parity.	**None** (Vendor Limitation)
Data_Bits*	Specify data bits.	**8** (Vendor Limitation)
Stop_Bits*	Specify stop bits.	**1** (Vendor Limitation)
Poll _Delay*	This commonly used parameter has no meaning for this driver.	

Example

```
//  Client Side Connections

Connections
Port    , Protocol     , Baud    , Parity
P1      , FCI_7200     , 1200    , None
```

[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.

5.2 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 commonly used parameter has no direct meaning for this driver.	The protocol is node-less. This means the messages do not contain information identifying a unique source and/or destination node address. For this reason this parameter needs not be specified and only one SCU can be connected per FieldServer serial port.
Protocol	Specify protocol used.	FCI_7200, FCI_7200_series, FCI7200
Connection	Specify which port the device is connected to the FieldServer.	P1-P2, R1-R2[2]

Example

```
//    Client Side Nodes

Nodes
Node_Name    , Protocol      , Connection
Panel-01     , FCI_7200      , P1
```

5.3 Client Side Map Descriptors

5.3.1 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 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 Client Map Descriptor. Passive and Passive_Client can be used interchangeably. Passive_Client is recommended unless there is an active Server Side on the other protocol in which case, Passive must be used.	Passive_Client, Passive

[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.

5.3.2 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**
Event_Type*	This driver uses this parameter to determine the suitability of a Map Descriptor when looking for a location to store data from an incoming message. A Map Descriptor may be defined to store only 'Alarm', 'Fault', 'Trouble' or 'Other events. Alternatively, specify "Any". A table of events vs. categories is provided in **Appendix B.1**.	**Any**, Other, Fault, Alarm, Trouble
Point_Type	This driver uses this parameter to determine the suitability of a Map Descriptor when looking for a location to store data from an incoming message.	Zone, Relay, Loop, Sensor, Module, Panel
Relay/Loop/Zone_Number	Ignored when the Point Type is 'Panel' Point Type = Relay 1-4 Point Type = Zone 1-8 Point Type = Loop 1-2 Point Type = Module 1-2 Point Type = Sensor 1-2	Positive Intergers
Length	Each Map Descriptor defines storage locations for a series of addresses. This parameter specifies the length of the series.	Positive Intergers
Address*	This parameter is only considered for those Map Descriptors whose 'Event Type' is Module or Sensor. It specifies the starting module or sensor number. The length parameter the determines the range of the sensor/module numbers.	1 to 99, **-**
Store_As*	Set this parameter to 'Bit' to have the driver use the primary Data Array to store using the 'Bit Storage' Method. These methods are described in **Appendix B.2**.	Bit, **Index_Value**
DA_Bit_Name*	If the default 'Store As' is specified or if the parameter is omitted then it is an option to specify a secondary array using this parameter - the driver will store event data as 'Bit Storage' in the secondary array (and as 'Index Values' in the primary array.) These methods are described in **Appendix B.2**.	One of the Data Array names from **Section 4**
Clear_On_Reset*	If this parameter is configured and set to "No", data will not be cleared if a "reset" message is received by the panel.	**Yes**, No

5.4 Map Descriptor Example 1 – Panel Events

In this example a map descriptor defines storage locations for messages sent by the panel containing panel events. A panel event is one that is not a relay, loop, zone, sensor or module. Includes events such as battery failure, resets, silences etc. As these messages do not contain any address information all data will be stored at a single location and the length should be 1. Each time a new event is received the driver stores a value indicating the event cause.

FAULT: AC Power SCU 0:00:04 1/01/92

```
//    Client Side Map Descriptors

Map_Descriptors
Map_Descriptor_Name   , Data_Array_Name   , Data_Array_Offset   , Function          , Node_Name   , Event_Type
PanelData             , DA_PANEL          , 0                  , Passive_Client    , Panel-01    , Any
```

```
, Point_Type   , Address   , Length   , Clear_On_Reset
, Panel        , -         , 1        , Yes
```

Example comments:

- Map_Descriptor_Name – It is recommended to allocate unique MD names.
- Data_Array_Name & Data_Array_Offset – Tell the driver the Data Array name and starting location that data should be stored.
- Function – The driver listens passively for messages from the Panel. It cannot poll for data.
- Node_Name – The name of the Node defined in the Node Descriptor section.
- Event_Type – The driver will not apply a filter to incoming events. Any event qualifies for storage.
- Point_Type – Only ‘Panel’ events will be stored using this MD.
- Length – The driver will clear the 1(=Length) element of the DA called DA_Panel starting at offset=0 when a Reset message is received.

5.5 Map Descriptor Example 2 – Sensor / Module Events

If messages from Loop 1, Module 1 to 99 are received then the MD (in this example) will be used for storage. If there are modules on more than one loop then one MD for each loop must be used. In the following example, the event type is set to 'Alarm'. This means that only 'Alarm' events will be stored using the MD. This method is useful to capture a single category of events. To store all events, change the 'Event Type' to 'Any'.

EEPROM BAD: QZUb L1M01 << Chief's Office >> 5:24 3/03/93

```
//   Client Side Map Descriptors

Map_Descriptors
Map_Descriptor_Name   , Data_Array_Name   , Data_Array_Offset   , Function         , Node_Name   , Event_Type
ModuleData1           , DA_MODULE         , 0                   , Passive_Client   , Panel-01    , Alarm
```

```
, Point_Type   , Relay/Loop/Zone_Number   , Address   , Length   , Clear_On_Reset
, Module       , 1                        , 1         , 99       , Yes
```

Example comments:

- Event_Type – In this example, only Alarm events will be stored. Messages reporting other events will be ignored unless other Map Descriptors are defined.
- Point_Type – Change this to 'Sensor' for sensors.
- Address & Length – The address specifies the stating Module number and the Length tells the driver the range of Modules. In this example. Module 1 to 99.

5.6 Map Descriptor Example 3 – Bit Storage

This example defines storage location for Relay Point events. The example would work for all other point types. In the example, both primary and secondary storage Data Arrays have been specified. The driver stores index values in the primary array. Each new event for a particular relay will overwrite the value stored previously. In the Bit Array, the driver sets the bit corresponding to the event, leaving other bits unchanged – thus the 2ndary storage can be used to determine if more than one event is active at a time.

```
//   Client Side Map Descriptors

Map_Descriptors
Map_Descriptor_Name   , Data_Array_Name   . Data_Array_Offset   . DA_Bit_Name   . Function         . Node_Name
RelayData             . DA_RELAY          . 0                   . DB_Relay      . Passive_Client   . Panel-01
```

```
. Event_Type   . Point_Type   . Relay/Loop/Zone_Number   . Address   . Length   . Clear_On_Reset
. Any          . Relay        . 1                       . -         . 4        . Yes
```

Example comments:

- Data_Array_Name – Where the primary DA is specified. Index values are stored here.
- DA_Bit_Name – where secondary storage is defined. Events are stored by setting appropriate bits. Remember that 2 elements per Relay, Module, Sensor, Loop are used.
- Address – Map Descriptors for storing Relay, Loop, Zone and Panel do not need the address specified.

Appendix A. Useful Features

Appendix A.1. Extending the Event Table

New event causes may be added to the Event Table (**Appendix B.1**) and the index value or category of existing event causes modified by adding a section to the configuration CSV file. The examples below illustrate this.

Appendix A.1.1. Map Descriptor Parameters

Column Title	Function	Legal Values
Event_Type_Description*	Name (Description) of the new Event Types	Any string – max 19 characters
Event_Type_Index_Value	Provide the value that should be stored for a specific event.	-32768 to 32768, **0**
Event_Type_Category	Specify the category to which the new event should belong	-32768 to 32768, **0**

Appendix A.1.2. Example 1 – Index Value "Trouble" Updated to a Value of 100

```
Driver_Table
Event_Type_Description   , Event_Type_Index_Value   , Event_Type_Category   , Protocol
TROUBLE                  , 100                      , 4                     , FCI_7200
```

Appendix A.1.3. Example 2 – New Entry Added

Since it has been added as category=3, only Map Descriptors with 'Event Type' set to Alarm or ANY will capture messages with this event description.

```
Driver_Table
Event_Type_Description   , Event_Type_Index_Value   , Event_Type_Category   , Protocol
DESTROYED                , 51                       , 3                     , FCI_7200
```

For categories use the following values:
'Other' = 1
'Fault' = 2
'Alarm' = 3
'Trouble' = 4

Appendix A.2. Panel Synchronization

Manual synchronization is required. Push the reset button on the panel. This transmits a reset message to the FieldServer, which clears the data in the FieldServer. After a reset the panel sends messages to report all abnormal states. When all these messages have been processed the FieldServer and panel will be synchronized. This process can be repeated at any time.

Appendix B. Reference

Appendix B.1. Events and Event Categories

The driver reports the event cause using the matching index value. There are 4 event categories:

1 = Other
2 = Fault
3 = Alarm
4 = Trouble

The message category must match the 'Event Type' parameter specified on a Map Descriptor before that Map Descriptor can be considered for storage of the message data.

Index	Category	Event
1	2	"Fault"
2	1	"Short"
3	1	"Disconnect"
4	1	"Comm Fault"
5	1	"Config Err"
6	1	"Eeprom Bad"
7	1	"Reset"
8	1	"Silence"
9	1	"Cross Zone"
10	1	"Acknwldgd"
11	1	"Walk Test"
12	1	"Alarm Test"
13	1	"SPVSN Test"
14	1	"Fault Test"
15	1	"Fire Drill"
16	1	"Batt Test"
17	1	"PRGM Mode"
18	1	"Action"

Index	Category	Event
19	1	"Loop Break"
20	3	"Alarm"
21	1	"P.A.S."
22	1	"Off-Normal"
23	1	"RZA Fault"
24	1	"Verify"
25	1	"CM SHort"
26	1	"Test Fail"
27	1	"Alert"
28	1	"Dirty"
29	1	"Very Dirty"
30	1	"Missing"
31	1	"Wrong Type"
32	1	"Extra Addr"
33	1	"Clock Err"
34	4	"Trouble"
35	1	"MLT Events"
36	1	"Alrm Ackd"

Appendix B.2. Data Storage

All messages less than 102 characters long are discarded. All other messages are processed as follows:

- The driver determines if the message is a Zone, Relay, Loop, Sensor, Module or Panel message.
- The driver finds all Map Descriptors with matching ‘Point Type’ parameters.
- The event category is determined.
- Map Descriptor selection is refined according to the ‘Event Type’ specification.
- The driver determines the Loop. Relay, Zone, Sensor and Module numbers from the message and refines its selection of Map Descriptors by selecting those that match the values determined from the message.
- The selected Map Descriptors are now used to determine a data array and offset at which to store the data.
- Finally the driver checks the ‘Store As’ parameter. If it hasn’t been specified then ‘Index Value’ storage is assumed. If it has been specified as ‘Bits’ then the driver will perform ‘Bit Storage’. In cases where the Map Descriptor has both a primary and secondary Data Array, the driver will use ‘Index Value’ storage using the primary data array and ‘Bit Storage’ using the secondary array.

Example:

The following fragment is part of a Map Descriptor definition; some parameters have been omitted for the purposes of clarity.

```
Map_Descriptors
Data_Array_Name , Data_Array_Offset , Event_Type , Point_Type , Relay/Loop/Zone_Number , Address , Length , Clear_On_Reset , DA_Bit_Name
DA_MODU         , 0                 , Any        , Module     , 1                       , 1       , 99     , Yes            , DB_MODU
DA_MODU_A       , 0                 , Alarm      , Module     , 1                       , 1       , 99     , Yes            , DB_MODU_A
DA_MODU_F       , 0                 , Fault      , Module     , 1                       , 1       , 99     , Yes            , DB_MODU_F
DA_MODU_T       , 0                 , Trouble    , Module     , 1                       , 1       , 99     , Yes            , DB_MODU_T
DA_MPODU_O      , 0                 , Other      , Module     , 1                       , 1       , 99     , Yes            , DB_MODU_O
```

Message = “FAULT: AC Power SCU 0:00:04 1/01/92”

- This message does not report the status of a Zone, Relay, Loop, Sensor or Module and is therefore assumed to be a panel message. Since there is no Map Descriptor with “Point Type” Panel, the message is ignored.

Message = “TROUBLE: QZUb L1M22 << Chief's Office >> 5:24:00 3/03/93”

- This message reports status for Loop 1 Module 22. Since all the Map Descriptors in the example have a ‘Point Type’=‘Module’, they are all considered for storage.
- The driver looks in the Event Table and finds it has an index value of 34 and a category of 4 (Trouble). Only the Map Descriptors with “Event Type” set to “Any” and “Trouble” are now considered.
- Since the value of the ‘Relay/Loop/Zone’ parameter matches the Loop number in the message, these Map Descriptors remain in contention.
- The Module number of 22 is compared with the Map Descriptors Address and Length Parameters. The Address is the starting number and the length defines the range. Both Map Descriptors have addresses of 1 and length of 99 and thus both are still selected because the Module of 22 falls in this range.
- The driver calculates an offset based on the offset specified in the MD and the Module number relative to the Map Descriptors address:
  - MD Offset = 0
  - MD Address = 1
  - Message Module = 22
- Module 1’s data is stored at offset 0 and hence Module 22’s data will be stored at offset 21. The driver stores the value 34 at offset 21 overwriting any data previously stored at that location. This is ‘Index Value’ Storage.
- Secondary storage has been defined using the ‘DA_Bit_Name’ Data Array. The driver doubles the offset as two locations are used for each address. Then the driver reads the value found in the Data_Array, modifies it and writes it back. As the index value is 34 the driver modifies the 34th bit – or expressed another way, the driver modifies the 2nd bit (34-32) at offset+1.
- Thus, driver calculates the offset for Bit Storage as 2 x 21 = 42. The driver sees that bit 34 is 2 2nd bit in the next offset and so the driver reads DB_MODU:43, modifies the value by setting the 2nd bit on and then writing the modified value back. During the modification all other bits are left intact. Thus using the Bit Storage method, a single Module (or sensor) can keep track of multiple events.