



---

A Sierra Monitor Company

**Driver Manual**  
**(Supplement to the FieldServer Instruction**  
**Manual)**

**FS-8700-83 Gamewell Serial Driver**

**APPLICABILITY & EFFECTIVITY**

**Effective for all systems manufactured after May 1, 2001**

## TABLE OF CONTENTS

<b>1. GAMEWELL SERIAL DRIVER DESCRIPTION .....</b>	<b>1</b>
<b>2. DRIVER SCOPE OF SUPPLY .....</b>	<b>1</b>
2.1 SUPPLIED BY FIELD SERVER TECHNOLOGIES FOR THIS DRIVER .....	1
2.2 PROVIDED BY USER .....	1
<b>3. HARDWARE CONNECTIONS .....</b>	<b>2</b>
<b>4. CONFIGURING THE FIELD SERVER AS A GAMEWELL SERIAL DRIVER CLIENT .....</b>	<b>3</b>
4.1 DATA ARRAYS .....	3
4.2 CLIENT SIDE CONNECTIONS .....	4
4.3 CLIENT SIDE NODES .....	5
4.4 CLIENT SIDE MAP DESCRIPTORS .....	6
<i>FieldServer Related Map Descriptor Parameters</i> .....	6
<i>Driver Related Map Descriptor Parameters</i> .....	6
<i>Timing Parameters</i> .....	7
<i>Map Descriptor Example 1 – Store data from incoming messages</i> .....	8
<i>Map Descriptor Example 2 – Send a Reset / Ack / Silence Command</i> .....	9
<b>5. CONFIGURING THE FIELD SERVER AS A GAMEWELL SERIAL DRIVER SERVER .....</b>	<b>10</b>
<b>6. ADVANCED TOPICS .....</b>	<b>10</b>
6.1 DRIVER LIMITATIONS & EXCLUSIONS .....	10
6.2 DATA TYPES .....	10
6.3 STATUS TYPES AND VALUES .....	11
<i>Adding a New Status Type</i> .....	12
6.4 WHEN DO DATA ARRAYS GET CLEARED (RESET) .....	12
6.5 ACTION TYPES .....	12
<i>Adding new Action Types</i> .....	15
6.6 ADVANCED MAP DESCRIPTOR EXAMPLES .....	16
<i>Example 1 : Filtering Data</i> .....	16
<i>Example 2 : Action Numbers</i> .....	17
<i>Example 3 : Action Bits</i> .....	18
<i>Example 4 : Ignored Messages</i> .....	19
<b>7. DRIVER NOTES .....</b>	<b>20</b>
7.1 IC_TIMEOUT .....	20
7.2 DRIVER STATS .....	20
<b>8. REVISION HISTORY .....</b>	<b>ERROR! BOOKMARK NOT DEFINED.</b>

## 1. Gamewell Serial Driver Description

The Gamewell Serial Driver allows the FieldServer to accept messages generated by a Gamewell 600 Series Panel as well as 'Smartnet Data Stream' messages generated by a Gamewell Smartnet terminal.

All Gamewell 600 Series Fire Alarm panels are equipped with a serial port, which produces panel, circuit or device status messages. This driver is designed to process these messages and store this status information in numeric form. The numeric value will indicate the type of event being reported and the storage location in the FieldServer's data arrays is (configurable &) dependent on the origin of the message (panel / circuit / device). Additional information such as event date and time and descriptions are ignored.

The driver is capable of supporting a panel configured to supervise the port by responding to the panel's supervision queries.

This is a passive client driver. The driver listens passively for unsolicited messages produced by the Gamewell panel. This definition is not strictly true because the driver is capable of sending the panel three messages: Ack, Silence and Reset.

Design Basis: Gamewell serial port protocol specification "IF 600r7 Message Stream" (not dated) and "SmartNet Data stream information" (not dated).

The driver is capable of exposing communication statistics in a FieldServer Data Array so that a remote device can monitor them.

## 2. Driver Scope of Supply

### 2.1 Supplied by FieldServer Technologies for this driver

FieldServer Technologies PART #	DESCRIPTION
FS-8917-16	RS-485/RS-232 Pigtail for RJ45 port
	Driver Manual.

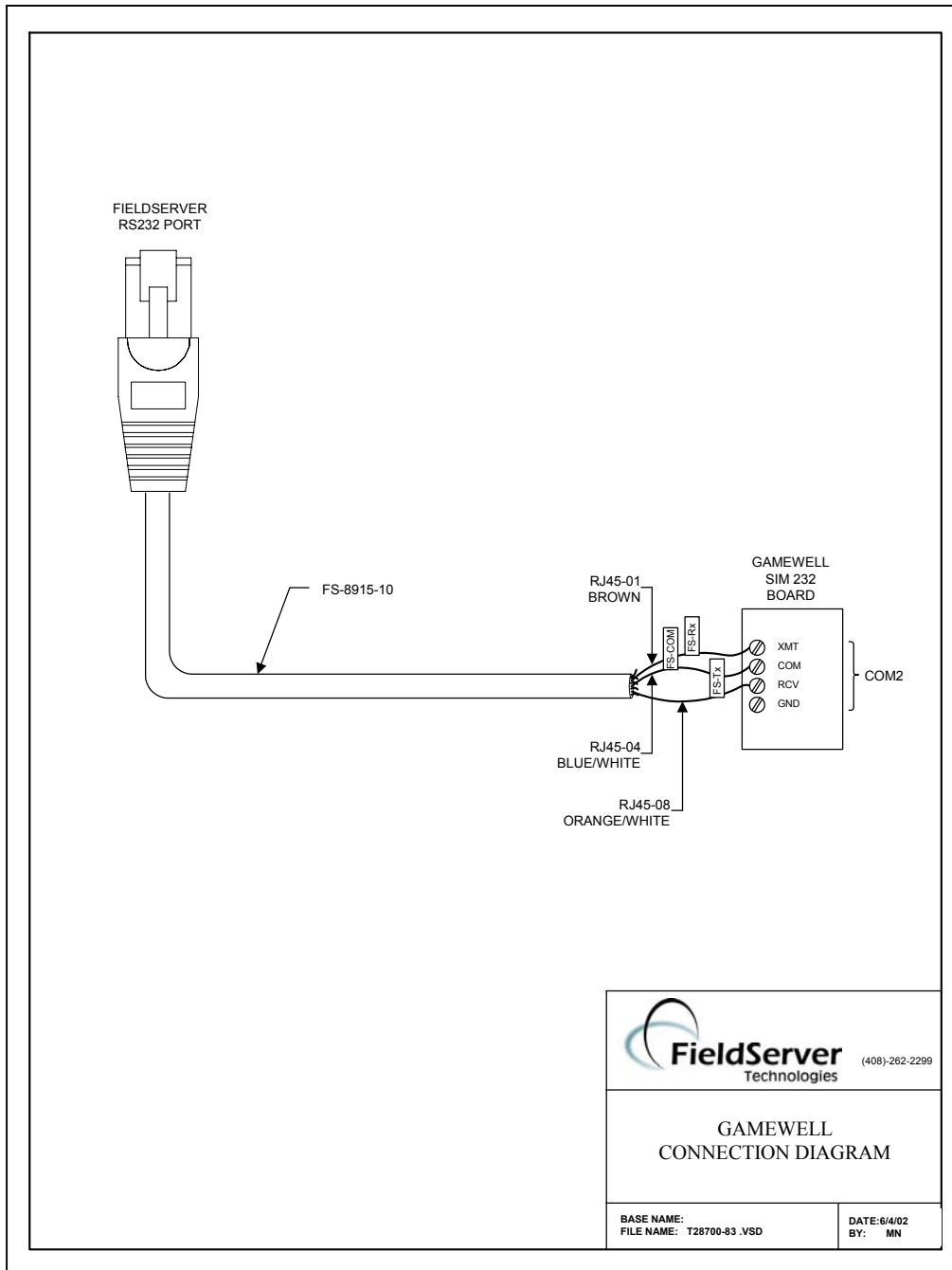
### 2.2 Provided by user

PART #	DESCRIPTION
	Gamewell Panel with SIM232 interface

**3. Hardware Connections**

The bridge is connected to the Gamewell panel as shown below.

Configure the Gamewell panel according to manufacturer's instructions



#### 4. Configuring the FieldServer as a Gamewell Serial Driver 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” files on the driver diskette).

This section documents and describes the parameters necessary for configuring the FieldServer to communicate with a Gamewell Serial Driver Server

The configuration file tells the FieldServer about its interfaces, and the routing of data required. In order to enable the FieldServer for Gamewell 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.

#### 4.1 Data Arrays

Section Title		
Data_Arrays		
Column Title	Function	Legal Values
Data_Array_Name	Provide name for Data Array	Up to 15 alphanumeric characters
Data_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 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 Connections

Section Title		
Connections		
Column Title	Function	Legal Values
Port	Specify which port the device is connected to the Bridge	P1-P8, R1-R2
Baud	Specify baud rate  The driver supports all standard baud rates 110 – 115200. Gamewell panels only support a baud rate of 2400.	2400
Parity*	Specify parity  The driver supports the following options. Even, Odd, <b>None</b> , Mark, Space  The Gamewell panels only support the use of no parity.	<b>None</b>
Data_Bits*	Specify data bits	<b>8</b>
Stop_Bits*	Specify stop bits	<b>1</b>
Protocol	Specify protocol used	Gamewell
Handshaking*	Specify hardware handshaking	<b>None</b>
Poll_Delay*	Time between internal polls	0-32000 seconds <b>default 1 second</b>
IC_Timeout	This driver does not use an inter character timeout system. Failing to set the IC_Timeout to zero will result in driver errors.	0

### Example

```
// Client Side Connections

Connections
Port, Baud, Parity, Protocol, Handshaking, Poll_Delay, IC_Timeout
P8, 9600, None, Gamewell, None , 0.100s , 0
```

### 4.3 Client Side Nodes

Section Title		
Nodes		
Column Title	Function	Legal Values
Node_Name	Provide name for node	Up to 32 alphanumeric characters
Node_ID	<p>Gamewell panel node address.</p> <p>The Node Id has no meaning when the FieldServer is connected directly to Gamewell panel. When connecting to a Gamewell panel directly always set the Node_ID to zero. When connecting to a Gamewell SmartNet terminal. The Node_ID is important and should correspond to the Node_ID's of the panel's connected to the SmartNet terminal.</p>	0-256
Protocol	Specify protocol used	Gamewell
Port	<p>Specify which port the device is connected to the FieldServer.</p> <p>The use of R1 and R2 is only appropriate when a 232/485 converter is used. The Gamewell panels only have a 232 port available for connection.</p>	P1-P8, R1-R2

**Example**

```
// Client Side Nodes

Nodes
Node_Name, Node_ID, Protocol, Port
Panel1 , 0 , Gamewell, P8
```

#### 4.4 Client Side Map Descriptors

##### 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 Bridge	One of the Data Array names from "Data Array" section above
Data_Array_Location	Starting location in Data Array	0 to maximum specified in "Data Array" section above
Function	Function of Client Map Descriptor.  Reads (rdbc / rdb ) are not allowed. The use of WRBX is recommended for the ack / silence / reset functions. A message will be generated each time the value in the associated array is updated (even if the value stays the same.)	Passive, WRBC, WRBX

##### 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	Data type  This parameter is only required for passive / server map descriptors.  The Data Type determines the type of data that gets stored when a message get received. The Data Type corresponds to the 'Status' field in a Gamewell message.  Additional information is provided in section 6.2	Any Alarms Faults Events Bus Comm Control Ack Signal Silence Troubles Supervisories Action_Numbers Action_Bits Dump



Length	Length of Map Descriptor  Controls how many elements of the data array are controlled by the map descriptor.	1 – 1000
Address	This commonly used parameter has no meaning for this driver. If specified it is best set to zero.	0
Ckt	Specify the circuit number whose message will get stored using this map descriptor.  Use the keyword 'Panel' if the you wish to store data from a panel. All messages which do not contain the keyword 'Ckt' in the action field are deemed to be panel messages.  Valid panel numbers are 1-131	Panel , 1 ,2 , 3 .... 131
Dev	The starting device number for the map descriptor. The length determines how many devices can have their data stored using this map descriptor.  Valid circuit numbers are 1-126 but this driver allows a device number of zero to allow for the storage of messages which don't specify a device number.  Ty	0 , 1 , 2 ... 126
Clear_On_Reset		Yes, <b>No</b>
Store_As*	Only relevant when the Data_Type='Dump'  This tells the driver to store ignored messages in ASCII format or to dump them in ASCII format in the error log.	AsciiLog ASCII <b>Value</b>
Gamewell_Func*	Only relevant when the function is write.  This parameter tell the driver what type of command to send to the panel.  You should always Ack before you silence the panel.	Reset Ack Silence

Timing Parameters

Column Title	Function	Legal Values
Scan_Interval	Rate at which data is polled	>0.1s

Map Descriptor Example 1 – Store data from incoming messages.

*This example illustrates typical map descriptor uses to store data from panel generated messages.*

*As all these map descriptors have their Clear\_On\_Reset field set to yes, when a panel reset message is received all the data in the controlled arrays will be set to zero. When a point reports its own state as normal the driver will set the appropriate element of the appropriate array to zero to indicate the normal state..*

*Data is stored, first by finding a map descriptor with the correct circuit number. If the message doesn't contain a circuit number then it is assumed to be from the panel itself. If a message contain a CKT number and no DEV number then the driver assumes the device number is zero. The storage location is based on the device number – it is used as an offset into the array.*

```
Map_Descriptors
Map_Descriptor_Name, Data_Array_Name, Data_Array_Offset, Function, Node_Name, ckt , Dev, Length, Data_Type , Clear_On_Reset
Panel_data , DA_STATUS , 000 , passive , panell , Panel, 0 , 100 , Any , Yes
Ckt1_data01 , DA_STATUS , 200 , passive , panell , 1 , 0 , 100 , Any , Yes
Ckt2_data01 , DA_STATUS , 400 , passive , panell , 2 , 0 , 100 , Any , Yes
Ckt3_data01 , DA_STATUS , 600 , passive , panell , 3 , 0 , 100 , Any , Yes
```

In this example all the data for all these circuits is stored in one array.  
  
The offset is used to control the location in the array.

These map descriptors are all passive. We cannot poll the panel but we can wait passively for the panel to send us messages.

You need one map descriptor for each circuit / panel.

The length determines the number of devices that can be processed using the map descriptor.  
  
Say a message for Ckt:2 Dev:20 is received. The driver looks at the device number and the length to see if the range of devices covers the incoming message. In this case the data would be stored at offset 20 in the map descriptor.

Because the Data Type is 'Any' the driver will set an array element non-zero if any messages indicate that the point is not in a normal condition.

Map Descriptor Example 2 – Send a Reset / Ack / Silence Command.

*This example illustrates three map descriptors used to send commands to the panel. These are the only active map descriptors that can be used with the Gamewell Serial Driver.*

*These map descriptors use the WRBX function. When the 1<sup>st</sup> element (because Data\_Array\_Offset = 0 ) has its value updated, even if the value doesn't change, then the driver will send the command to the panel.*

*Note that you are required to send an Ack before you can send a silence command. The driver does not clear the trigger by setting the array element back to zero. The panel does not send a message acknowledging receipt of the command, thus the driver cannot provide positive confirmation.*

Map\_Descriptors

Map_Descriptor_Name	Data_Array_Name	Data_Array_Offset	Length	Function	Node_Name	Gamewell_Func
Ack_md	DA_ACK	0	1	wrbx	panell	Ack
Sil_md	DA_SILENCE	0	1	wrbx	panell	Silence
Res_md	DA_RESET	0	1	wrbx	panell	Reset

By using a wrbx you can have the driver send the command when the array is updated.

Thus, to trigger any of these commands have the remote device send a value to the 1<sup>st</sup> element of the above data arrays.

Use one of these keywords.

## 5. Configuring the FieldServer as a Gamewell Serial Driver Server

The Gamewell Serial Driver provides limited server functionality. This has been developed to allow for automated testing and Quality Assurance. It is not supported or documented however, at a client's request it can be extended & documented (typically at an additional cost.)

## 6. Advanced Topics

### 6.1 Driver Limitations & Exclusions

The driver does not support scaling when data is stored in a data array. The keywords

*'Data\_Array\_Low\_Scale, Data\_Array\_High\_Scale, Device\_Low\_Scale, Device\_High\_Scale'*

have no meaning for this driver. The reason for this is that the values stored by the driver have specific meanings based on parsing the message. Scaling is only applicable in drivers which read and write values from the remote device.

### 6.2 Data Types

Messages contain Status and Action information. The status information indicates the state of a device / circuit / panel. The action information describes the event that generated the message.

By specifying one of the following Data\_Types you can filter the incoming messages so that only certain types of messages update certain data arrays. For example, if you are only interested in storing data from messages that report an alarm then set the Data\_Type of that map descriptor to 'Alarms'. If you don't care about the particular state then use the Data\_Type of 'Any'. The driver will set the values of the array elements non-zero if any non-normal states are reported.

Data_Type	Status	Note #
Any		1
Alarms	Status:ALARM	2
Faults	Status:FAULT	2
Events	Status:EVENT	2
Bus	Status:BUS	2
Comm	Status:COMM	2
Control	Status:CONTROL	2
Ack	Status:ACK	2
Signal Silence	Status:SIG SIL	2
Troubles	Status:FAULT	2
Supervisories	Status:EVENT and Action contains "Supv. Event in"	2
Action_Numbers		3
Action_Bits		4
Dump		5

Notes:

1. If the Data Type is 'Any' then the map descriptor will be used to store data from message with any status.
2. The state reported is filtered and must match the Data\_Type for the associated array to be updated. For example, if the Data\_Type of a MapDesc is 'Alarms' and a message is received that reports a Fault then the map descriptor will not be used to store the data from the message.
3. Normally, user's are interested in the Status of a device / circuit / panel but they may also be interested in the cause (or the 'action' in Gamewell terminology) of the message. When you specify the Data\_Type as 'Action Numbers' then the driver will store a value which can be used to look up the action that produced the message. The most recent action number is stored over any older value. (The driver does not provide an event log.)
4. Instead of storing a value to indicate the action, the driver can set a bit whose offset indicates the action. For example, action 30 will cause the 30<sup>th</sup> bit to be set. Action bits are stored retentively. This means that when a new action is reported the previous bits are left set and a new bit is set too.
5. What happens if a message arrives that reports an alarm and you don't have a map descriptor with a Data\_Type capable of storing an alarm. You can make a catch all map descriptor and use the 'Dump' Data\_Type to tell the driver to store the whole message in ASCII format in a data array so that you can inspect it. You can also use this map descriptor to tell the driver to dump the ignored message to the error log.

### 6.3 Status Types and Values

Generally, the driver stores non-zero values to indicate the state of a device / circuit / panel based on the 'status' field of the incoming message. The specific non-zero value can be found in the following table. The value have been chosen so that they correspond to different bits.

Status	Value Stored
Alarm	1
Fault	2
Event	4
Bus	8
Comm	16
Control	32
Ack	64
Sig Sil	128
Supv	512
Genr	1024

Thus if a map descriptor has its Data\_Type = 'Any' and two messages are received , one an alarm and one an event, then the value of the array element will be set to  $1 + 4 = 5$ . Thus the value is non-zero to report the not-normal state but inspection of the value allows you to determine the specific stat.

The value's can be changed by using the method below to add, but when you add, use the existing name and a new value.

When comparing these keywords to the data in the Status field of the message the driver only compares the first three characters. The comparison is case insensitive.

### Adding a New Status Type

The fragment of a CSV file displayed below illustrates how to change the value associated with 'BUS' to 9 and adds two new Status types, Fred and Ginger.

```
Driver_Table
Gamewell_Status_String, Gamewell_Status_Value, Protocol
BUS, 9, Gamewell
FRED, 100, Gamewell
GINGER, 101, Gamewell
```

There is a limitation in the use of new status types. They can only be stored using map descriptors with the Data\_Type set to 'Any'.

The driver can store a maximum of 100 status types. The maximum length of the string is 9 characters.

## 6.4 When do Data Arrays get cleared (Reset)

When the Gamewell Panel is reset then the driver is able to clear the arrays. The way that the panel works is that when a reset is performed send the following message

```
Status:NORMAL          08/31/95 16:23
System Idle
```

After this message the panel then sends messages for all points that are not in a normal state.

The driver uses the parameter 'Clear\_On\_Reset' to determine what gets cleared. If a map descriptor has this parameter set to 'Yes' then the arrays elements controlled by the Data\_Array\_Offset and the Length are set to zero.

This provides a good technique of synchronizing the panel and the FieldServer. When you restart the FieldServer you should push the reset button on the panel so that the panel sends messages for all points that are not in a normal state. If you don't do this, and some points are in a not-normal state then the FieldServer will not know about them until their state changes This is potentially dangerous.

## 6.5 Action Types

Typically messages from the panel contain not only the status of a point but also describe the action that caused the state to change.

Based on the table below if a message contains the string 'Fire Alarm in' then the action number will be stored as 11.

<b>Value Action</b>	<b>Description</b>
255	Driver did not recognize action type
<b>System Setup</b>	
1 Skip System I/O Assignments	System power up
2 Begin System I/O Assignments	System programming itself
3 Programming Mode Entered	Ignore any data from this point until "Exit Program Mode" is received
4 Exit Program Mode	See Programming Mode Entered
<b>Control</b>	
5 Commencing System Reset	System reset button has been depressed
6 System Idle	System reset completed system is normal
7 System Acknowledged	System Acknowledge button depressed
8 Signals Deactivated	System Signal Silence button depressed Audibles Silencing
9 Signals Activated	System Signal Silence button depressed Audibles reactivating
10 Signals Silenced Automatically	System automatically silenced the audible signals
<b>Fire Alarm</b>	
11 Fire Alarm in	
<b>Supervisory</b>	
12 Supv. Event in	
<b>Generic</b>	
13 Genr. Event in	
<b>Security Alarm</b>	
14 Security Alarm in	
<b>Pre Alarm</b>	
15 Ver. Seq. in	Verification sequence started
16 Pos Al. Seq. in	Positive Alarm Sequence started
17 Pre-Alarm in	Pre alarm present
<b>Fault</b>	
18 Alarm Tested in	Message during walk test
19 AtoD Malfunction	System Problem
20 LCD Malfunction	System Problem
22 System In Walk Test	Start partial or Full system walk test mode
23 System Out Of Walk Test	Finished system walk test mode
24 System I/Os By Passed	Bypass system circuits or devices
25 All By Passed I/Os Cleared	System not bypassed
26 I/O Bypassed,	Starts ID of Circuit or point bypassed
27 Remote Annunciators Not Responding	System Problem
28 Remote Annunciators OK	System Problem Restored
29 Key Stuck in	System Problem
30 Display Missing for	System Problem
31 Bad Card @	System Problem
32 Card Missing @	System Problem

33 New Card Detected @	System Problem
34 Out of Memory Assigning	System Problem
35 I/O Restored	Supervisory or trouble in circuit that automatically restore themselves
36 Trouble Tested in	System Problem
37 Trouble in	System Problem
38 Output Shorted in	System Problem
39 Dup. Dev. in	System Problem
40 Dev. Missing in	System Problem
41 Type Mismatch	System Problem
42 Dev. Dirty in	System Problem
43 No Response from Analog CKT	System Problem
44 Open/Short in CKT	System Problem
45 I/O Not Detected	System Problem
46 Password Accepted	System Valid password entered
47 +5V OK On	System Problem
48 +5V Bad On	System Problem
49 Aux. Supply OK For	System Problem
50 Aux. AC Bad For	System Problem
51 Aux. Batt. Bad For	System Problem
52 Aux. Bad For	System Problem
53 Unknown Event	System Problem
54 Communication Failure	Master lost communications with Node
55 Communication Restored	Master restored communications with Node X
54 Communications Failure	Master lost communications with Node
55 Communications Restored	Master restored communications with Node X
56 Primary Bus Error	Break or short in the primary class A cable
57 Secondary Bus Error	Break or short in the secondary class A cable
58 Printer fault	Master printer error
59 Batt. Charging OK	
60 Batt. Charging	



## Adding new Action Types

The following fragment from a CSV file shows how you can add two new action types. If a message is received and its action field contains the text 'FRED' then the action number will be stored as 100.

```
Driver_Table
Gamewell_Action_String, Gamewell_Action_Value, Protocol
FRED, 100, Gamewell
GINGER, 101, Gamewell
```

The driver can store a maximum of 100 action types. The maximum length of the string is 49 characters.

## 6.6 Advanced Map Descriptor Examples

### Example 1 : Filtering Data

You can direct the driver to filter the incoming messages so that data arrays are only updated for particular states. For example an incoming message which reports a device in Ckt 1 to be in a FAULT state (Status:FAULT) will use the map descriptor 'Ckt1\_data03' to store the data and the array DA\_FAULTS1 will be updated. If however, the message reported an ALARM state (Status:ALARM) then the array DA\_ALRMS1 would have been updated.

In fact, because the examples below provide a map descriptor where the data type is 'Any', each incoming message would update two data arrays. The DA\_Status1 array would be updated by every single message and the other arrays would be updated depending on the state being reported in the message.

Map\_Descriptors

Map_Descriptor_Name	Data_Array_Name	Data_Array_Offset	Function	node_name	ckt	Length	Data_Type	Clear_on_Reset
Ckt1_data01	, DA_STATUS1	, 000	, passive	, panell	, 1	, 100	, Any	, Yes
Ckt1_data02	, DA_ALARMS1	, 000	, passive	, panell	, 1	, 100	, Alarms	, Yes
Ckt1_data03	, DA_FAULTS1	, 000	, passive	, panell	, 1	, 100	, Faults	, Yes
Ckt1_data04	, DA_EVENTS1	, 000	, passive	, panell	, 1	, 100	, Events	, Yes
Ckt1_data05	, DA_BUS1	, 000	, passive	, panell	, 1	, 100	, Bus	, Yes
Ckt1_data06	, DA_COMM1	, 000	, passive	, panell	, 1	, 100	, Comm	, Yes
Ckt1_data07	, DA_CONTROL1	, 000	, passive	, panell	, 1	, 100	, Control	, Yes
Ckt1_data08	, DA_ACK1	, 000	, passive	, panell	, 1	, 100	, Ack	, Yes
Ckt1_data09	, DA_SIGSIL1	, 000	, passive	, panell	, 1	, 100	, Signal Silence	, Yes
Ckt1_data10	, DA_TROUBLES1	, 000	, passive	, panell	, 1	, 100	, Troubles	, Yes

You would need another set of these map descriptors for any other circuit you are monitoring.

By using specific data types, the driver will only update the associated data arrays when the messages report a state that matches the data type.

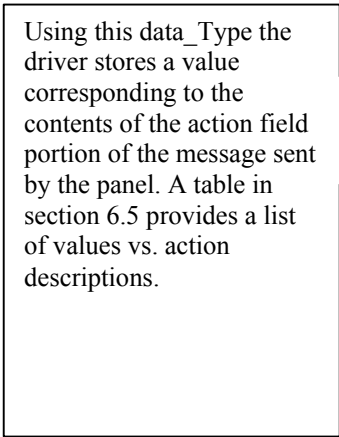
### Example 2 : Action Numbers

You can have the driver store a value corresponding to the contents of the action field reported in the incoming messages. Actions are brief descriptions of the event that caused the message to be generated. A table of values vs. descriptions is provided in section 6.5. The driver stores the most recent action number, overwriting the previously stored action numbers. The driver does not keep an event log. The action number's are set to zero, if the clear\_on\_reset is set to 'yes' and a system reset message is received.

For example, if the string 'Fire Alarm in' is contained in the action field of the message the driver would store an action value of 11.

This map descriptor can be used as well as the any of the map descriptors shown in previous examples. Thus you can have one (or more) map descriptor storing the state and one storing the action number.

```
Map_Descriptors
Map_Descriptor_Name, Data_Array_Name , Data_Array_Offset, Function, node_name, ckt , Length , Data_Type , Clear_on_Reset
Ckt1128_data12 , DA_ACTION128 , 000 , passive , panell , 128 , 127 , Action_Numbers , yes
```



Using this data\_Type the driver stores a value corresponding to the contents of the action field portion of the message sent by the panel. A table in section 6.5 provides a list of values vs. action descriptions.

Example 3 : Action Bits

Instead of having the driver store a value to indicate the action, the driver can set a bit, whose offset indicates the underlying action. For example, if the string 'Fire Alarm in' is contained in the action field of the message the driver would set the array element at offset 11 (use table 6.5 to get the value vs. string) to 1.

Important to note is that the driver does not clear a previously set bit when a new action is reported. Thus if two messages were received and the first reported 'Fire Alarm in' and the second reported 'Supv. Event in ' then first the array element at offset 11 would be set to 1 and then element at offset 12 would be set. The element at offset 11 would remain set. Both would then remain set until a system reset is performed, the state of the point returns to normal (status:NORMAL) or you clear the bits by writing to the array from the remote device.

As the driver may use up to 100 consecutive array locations for each Ckt/Device pair, if you use this method of storing data you will need one map descriptor for each Ckt/Device pair. In this example the map descriptor will store data for Ckt 128 device 10 only. This is indicated by the Ckt number being set to 128, the device number being set to 10 and the length being set to 1.

If the driver doesn't recognize the action type then it will set the array element at offset zero.

```
Map_Descriptors
Map_Descriptor_Name, Data_Array_Name , Data_Array_Offset, Function, Node_Name, Ckt, Dev, Length , Data_Type , Clear_On_Reset
Ckt1128_data12 , DA_ACTION_BITS , 000 , passive , pane11 , 128, 10 , 1 , Action_Bits , yes
```

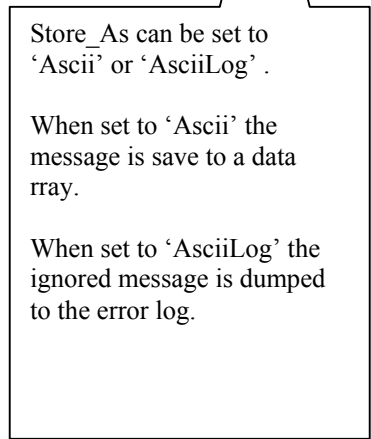
Only one device per map descriptor.

Ensure that at 100 elements of data array are available for each map descriptor. ( 100 is the maximum action number.)

### Example 4 : Ignored Messages

When messages are received that the driver cannot find a map descriptor to use to store the data from the message (say a message from a device on circuit 127 is received but there are no map descriptors for circuit 127) then the driver produces a MSG\_IGNORED stat. You can have the driver dump these messages to the error log or store the message in a data array by using the DATA\_Type='Dump'. If the data is stored in a data array then use a data array with a 'Byte' format and display the array using the ruinet utility and view the array in 'String' format. If you have the ignored messages dumped to the error log then use the RuiDebug utility to capture the error log

```
Map_Descriptor_Name, Data_Array_Name, Data_Array_Offset, Function, node_name, Length , Data_Type , Clear_on_Reset, Store_As  
Store_Ignored_Msg , DA_DUMP_IGNORED, 000 , passive , panell , 1000 , Dump , No , Ascii
```



Store\_As can be set to 'Ascii' or 'AsciiLog' .

When set to 'Ascii' the message is save to a data rray.

When set to 'AsciiLog' the ignored message is dumped to the error log.

## 7. Driver Notes

### 7.1 IC\_Timeout

The connection IC\_Timeout must be set to zero. This is done in the CSV file by setting IC\_Timeout parameter.

The following fragment from a CSV file illustrates how this is done.

```
Connections
Port, Baud, Parity, Data_Bits, Stop_Bits, IC_Timeout
P1, 2400, None, 8, 1, 0
```

### 7.2 Driver Stats

In addition to the standard FieldServer communication statistics described in the FieldServer User's Manual, the Gamewell Serial Driver can also expose some driver statistics by writing data to a data array. A special map descriptor is required. The driver recognizes the map descriptor by its name which must be "Gamewell-stats".

The following example shows how this special map descriptor can be configured. You can copy this section of text directly into your CSV file.

```
Nodes
Node_name, Node_ID, Protocol
dummy_node, 0, Gamewell

Data_Arrays
Data_Array_Name, Data_Format, Data_Array_Length
DA_GAMEWELL_STATS, uint16, 500

Map_Descriptors
Map_Descriptor_Name, Data_Array_Name, Data_Array_Offset, Function, node_name,
Gamewell-Stats, DA_GAMEWELL_STATS, 0, passive, dummy_node,
```

When the driver sees this map descriptor it uses the data array DA\_GAMEWELL\_STATS (in this example) to store driver specific statistics. Only one of these map descriptors may be specified per FieldServer.

The offset into the array is based on the port number. 30 arrays locations are used per port.

The offset is obtained by multiplying the port number by 30.

The driver stores the following data.

PORT				8	Description
0	1	2	3...		
<b>Array Offset</b>					
0	30	60	90	240	Available for future use
1	31	61	91	241	Available for future use
2	32	62	92	242	Available for future use
3	33	63	93	243	Available for future use
4	34	64	94	244	Number of bytes sent by client driver
5	35	65	95	245	Number of messages sent by client
6	36	66	96	246	Number of reponse messages received by client
7	37	67	97	247	Number of response bytes received by client
8	38	68	98	248	Number of times client has timeout out waiting for (response) prompt
9	39	69	99	249	Number of times client has timeout out waiting for (response) prompt
10	40	70	100	250	Number of Supervision Messages Sent
11	41	71	101	251	Number of Supervision Messages Received
12	42	72	102	252	Number of Supervision Messages Responses Sent
13	43	73	103	253	Number of Supervision Messages Responses Received
14	44	74	104	254	Number of Supervision Messages Received with a protocol error
15	45	75	105	255	Number of times that message containing status information was found
16	46	76	106	256	Number of times that message parsing failed because an unrecognized status type was found.
17	47	77	107	257	Number of time that a message containing node information was found
18	48	78	108	258	Number of times that oarsing failed because 2 CR's were not found.
19	49	79	109	259	Number of times that message parsing failed because an unrecognized status type was found.
20	50	80	110	260	Number of times that parsing completed
21	51	81	111	261	Number of times that the client timed out waiting for a message to be sent
22	52	82	112	262	Number of times that the client timed out waiting for a response to a supervision query
23	53	83	113	263	Number of times that a reset command was received.
24	54	84	114	264	Number of times that a Ack command was received.
25	55	85	115	265	Number of times that a Silence command was received.
26	56	86	116	266	Number of times that the slave received na unregonized command
27	57	87	117	267	Number of times that the slave received a command message
28	58	88	118	268	Number of times that the slave received a message

### 7.3 Simulating a Gamewell Panel

The driver provides support for QA procedures. One of the ways that the driver provides this support is by allowing a configuration to send a list of Gamewell messages to the driver. This allows a configuration developer or QA agent to test the effect of the messages on the driver – does the driver store the correct data in the correct locations etc.

1. Configure a client side MD as follows

```
Map_Descriptors
Map_Descriptor_Name, Scan_interval, Data_Array_Name, Data_Array_Offset, Function, Node_Name, Gamewell_Func
919sim.ini , 1.0s , DA_AI , 0 , wrbc , Node_1 , Simulator
```

The name of the file which contains the Gamewell messages.

Tells the driver to use the message file.

2. Create the simulation file.

This manual does not provide a list of possible messages. Customer logs or Gamewell manuals may be used.

The following is an example.

Each lines consists of 4 text segments encapsulated in angle braces “<>”. Gamewell messages consist of up to 4 lines of text. Put each line of the message into a <> segment. If the line begins with a # sign then it is ignored. Each time the MD becomes active (after its scan interval) the next line is read and sent. There is no method of recycling or repeating lines. For more examples of messages see the Word document attached to SPR2748.

```
<Status:ALARM I st of 1 08/31/95 16:25><Fire Alarm in Ckt:2><Fire Alarm Heat Detector><I st. Floor Room Number I>
<Status:FAULT 08/31/95 16:27><Aux. AC Bad For Ckt:2 Dev:1><><>
<Status:EVENT 08/31/95 16:26><Supv. Event in Ckt:2 Dev:2><Sprinkler Tamper Switch><I st. Floor Room Number I>
<Status:BUS 08/31/95 16:24><Supv. Event in Ckt:2 Dev:3><Sprinkler Tamper Switch><Ist. Floor Room Number 1>
<Status:COMM 08/31/95 16:25><I/O Restored, Ckt:2 Dev:4><><>
<Status:CONTROL 08/31/95 16:25><I/O Restored, Ckt:2 Dev:5><><>
<Status:ACK 08/31/95 16:25><I/O Restored, Ckt:2 Dev:6><><>
<Status:SIG SIL 08/31/95 16:25><I/O Restored, Ckt:2 Dev:7><><>
#<Status: NORMAL 08/31/95 16:28><System Idle><><>
```



3. Create a server side configuration
4. Create a script to test the server side configuration.

**8. Revision History**

Date	Driver Version	Document Revision	Resp	Comment
29May2002	1.00	0		Initial Release. Issued for review, formatting. Connection diagrams required.
13Jun2002	1.00	1		No Changes
19July2002	1.00	2		Releasing
18Sep2002	1.01	0		Added Status:SUPV support. Message that begin this way are stored as Supervisories (see section 6.3).  Changes to 6.2 Reference to Supervisory Data Type Changes to 6.3 Reference to the value of a SUPV message.
25Nov02	1.02	0		Changes to 6.3 Refrence to value of A GENR message Changes to 6.4 "Commencing System Reset" clears arrays too. Changes to 6.5 Variations for action types 13/16
20Aug03	1.02	1	JD	Releasing
6Sep03	1.02	2	PMC	Added 7.3. Notes on simulating a Gamewell Panel.