



A Sierra Monitor Company

ENOTE 0019

Squelch Handling

Rev 3

TABLE OF CONTENTS

1	Introduction.....	2
2	Setting Parameter Values.....	2
3	Statistics.....	3
4	Testing and Diagnostics.....	3

1 INTRODUCTION

Many half-duplex serial communication channels generate noise when the channel switches direction (typically at the end of a transmission burst), causing spurious data to be received at either end. The FieldServer kernel implements a user-configurable timing sequence designed to suppress the reception of this spurious data.

When the transmission ceases and the channel is released, noise can be generated at both the transmitting and receiving end. In a Master-Slave situation using poll and response messages this leads to four possible instances of squelch generation:

- squelch received by the master at the end of a poll transmission from Master to Slave
- squelch received by the slave at the end of a response transmission from Slave to Master
- squelch received by the slave at the end of a poll transmission from Master to Slave
- squelch received by the master at the end of a response transmission from Slave to Master

The first two are examples of what we will term TX squelch, received by the transmitting port at the end of a message, the last two are examples of RX squelch, received by the receiving port at the end of a message.

The timing diagram illustrates the four instances of squelch, and also identifies time intervals controlled by two connection parameters, i.e. `squelch_timer_tx` and `squelch_timer_rx`. These timers are activated at the appropriate moment, and for their duration prevent reception of data. The `squelch_timer_tx` starts at the end of a transmission (as defined by RTS becoming inactive), and the `squelch_timer_rx` starts at the end of a valid received message (as determined by the protocol driver). Note that the `squelch_timer_rx` is only relevant to slaves as masters will in any event disregard any spurious data received after a response.

2 SETTING PARAMETER VALUES

It is important to prevent the squelch suppression times from overlapping with valid data and interfering with proper communication. The following parameters can be configured in the FieldServer to address this.

Parameter	Description
Turnaround delay	(FieldServer connection parameter: <code>turnaround_delay</code>): this is the time the slave takes to initiate a response after having received a poll. The master connection must have a <code>squelch_timer_tx</code> value less than the turnaround delay.
Poll delay (FieldServer connection parameter: <code>poll_delay</code>):	this is the shortest time the master will wait between receiving a response message and initiating the next poll. The slave connection must have a <code>squelch_timer_tx</code> value less than the poll delay.

The following example depicts a connection statement

Connections					
Port	,Squelch_timer_tx	,Squelch_timer_rx	,Turnaround_delay	,Line_Drive_On	,Line_Drive_Off
p1	,0.06	,0.01	,0.050	,0.001	,0.001

3 STATISTICS

Each connection keeps track of the number of bytes suppressed as a result of tx and rx squelch timers. These may be viewed in the connection statistics screen.

4 TESTING AND DIAGNOSTICS

The Forth interpreter has been expanded to provide diagnostics that allow the driver developer to write scripts to simulate squelch and examine the resulting statistics. The relevant commands for triggering squelch are:

- `diag_squelch_mas_poll`
- `diag_squelch_mas_resp`
- `diag_squelch_slv_poll`
- `diag_squelch_slv_resp`

The connection squelch statistics may be retrieved with the following commands:

- `channel_2_stat stat_squelch_tx@`
- `channel_2_stat stat_squelch_rx@`

An example of the simulation syntax is given below:

```
//-----  
: do_comms_test  
  
  plc_tier  
  0 index_2_channel diag_squelch_mas_poll 1 submit_diagnostic  
  5 delay  
  
  scada_tier  
  0 index_2_channel channel_2_stat stat_squelch_tx@ 3 < ?line panic  
  0 index_2_channel diag_squelch_slv_poll 1 submit_diagnostic  
  5 delay  
  
  plc_tier  
  0 index_2_channel channel_2_stat stat_squelch_rx@ 3 < ?line panic  
  
  plc_tier  
  // not expecting anything strange here - master channel is cleared before next poll  
  0 index_2_channel diag_squelch_mas_resp 1 submit_diagnostic  
  3 delay  
  
  scada_tier  
  0 index_2_channel diag_squelch_slv_resp 1 submit_diagnostic  
  3 delay  
  
  plc_tier  
  0 index_2_channel channel_2_stat stat_squelch_tx@ 3 < ?line panic  
  ;  
//-----
```

Note that the driver must implement the generation of squelch. An example may be found in the Modbus RTU driver. The driver checks for diagnostic requests using the `do_diagnostic()` function. The diagnostic identification codes are defined in `<diag.h>`. For squelch testing the driver checks for:

- `DIAG_SQUELCH_MAS_POLL`
- `DIAG_SQUELCH_MAS_RESP`
- `DIAG_SQUELCH_SLV_POLL`
- `DIAG_SQUELCH_SLV_RESP`

The following code example shows how to implement the diagnostic:

```
/* this happens in the slave right after a complete poll has been received;
the characters are received by the master immediately after the poll,
simulating tx squelch */
if ( do_diagnostic ( &cmd->channel->channel_diagnostic, DIAG_SQUELCH_MAS_POLL ) )
{
    /* send a few spurious characters */
    CHAR *outbuf = "squelch";
    sys_printf("DIAG_SQUELCH_MAS_POLL");
    sioapi_puts ( cmd->channel->port_ptr->handle, (BYTE *)outbuf, 7 ) ;
}
```