FloBoss™ S600+ Flow Computer Modbus Specification Manual

Energy and Transportation Solutions

EMERSON™

Application & Device Safety Considerations

- ## Reading these Instructions
  Before operating a device or application, read these instructions carefully and understand their safety implications. In some situations, improper use may result in damage or injury. Keep this manual in a convenient location for future reference. Note that these instructions may not cover all details or variations in equipment or cover every possible situation regarding installation, operation, or maintenance. Should problems arise that are not covered sufficiently in the text, immediately contact Energy and Transportation Solutions (ETS) Customer Support for further information.

- ## Protecting Operating Processes
  The failure of a device or application – for whatever reason – may leave an operating process without appropriate protection and could result in possible damage to property or injury to persons. To protect against this, review the need for additional backup equipment or provide alternate means of protection (such as alarm devices, output limiting, fail-safe valves, relief valves, emergency shutoffs, emergency switches, etc.). Contact ETS for additional information.

- ## Using Qualified Personnel
  Installation, configuration, and any subsequent modifications to a device or application should only be performed by qualified, suitably trained personnel information.

- ## System Training
  A well-trained workforce is critical to the success of your operation. Knowing how to correctly install, configure, program, calibrate, and troubleshoot your Emerson equipment provides your engineers and technicians with the skills and confidence to optimize your investment. ETS offers a variety of ways for your personnel to acquire essential system expertise. Our full-time professional instructors can conduct classroom training at several of our corporate offices, at your site, or even at your regional Emerson office. You can also receive the same quality training via our live, interactive Emerson Virtual Classroom and save on travel costs. For our complete schedule and further information, contact the ETS Training Department at 800-338-8158 or email us at *education@emerson.com*.

- ## Grounding Equipment
  Ground metal enclosures and exposed metal parts of electrical instruments in accordance with relevant safety standards. For the USA, refer to OSHA rules and regulations as specified in *Design Safety Standards for Electrical Systems*, 29 CFR, Part 1910, Subpart S, dated: May 16, 1981 (OSHA rulings are in agreement with the National Electrical Code). For international locations, refer to IEC 60364-4-41: PROTECTION AGAINST ELECTRIC SHOCK. You must also ground mechanical or pneumatic instruments that include electrically operated devices such as lights, switches, relays, alarms, or chart drives. The chassis (or earth ground) lug provides a safe connection point to a customer-designated ground location for ESD and transient voltage suppression. Do not use the chassis ground lug for signal, common, or return connections. **Do not connect the chassis ground lug directly to a lightning arrestor/lightning rod.** Do not run signal wiring in conduit or open trays with power wiring or near heavy electrical equipment. If shielded wiring is used, ground the shield of the signal wiring at any one point of the signal loop.

  Important: Complying with the codes and regulations of authorities having jurisdiction is essential to ensuring personnel safety. The guidelines and recommendations in this manual are intended to meet or exceed applicable codes and regulations. If differences occur between this manual and the codes and regulations of authorities having jurisdiction, those codes and regulations must take precedence.

- ## Protecting from Electrostatic Discharge (ESD)
  Any device contains sensitive electronic components which can be damaged by exposure to an ESD voltage. Depending on the magnitude and duration of the ESD, it can result in erratic operation or complete failure of the equipment. Ensure that you correctly care for and handle ESD-sensitive components.

- ## Ethernet Connectivity
  This automation device is intended to be used in an Ethernet network which **does not** have public access. The inclusion of this device in a publicly accessible Ethernet-based network is **not recommended**.

- ## Returning Equipment
  If you need to return any equipment to ETS, it is your responsibility to ensure that the equipment has been cleaned to safe levels, as defined and/or determined by applicable federal, state and/or local law regulations or codes. You also agree to indemnify ETS and hold ETS harmless from any liability or damage which ETS may incur or suffer due to your failure to ensure device cleanliness.

Contents

Section 1. Introduction

This document describes the Modbus communications protocol on the FloBoss™ S600+ flow computer. The flow computer has 20 user-definable communication tasks. Each of the links may be individually configured.

Section 2. Physical Transport of Data

2.1 TCP Socket

The TCP/IP stream socket supports both master and slave links.

If the unit is the *slave* on the link, it creates a server socket and listens for clients to connect. The TCP/IP address of the slave link is the TCP/IP address of the flow computer.

If the unit is the *master* on the link, it acts as a client and attempts to connect to a remote server machine. When you configure the data link, you specify the TCP/IP address of the slave with which the master must communicate.

The FloBoss S600+ supports communication between networks using a gateway.

Other communication parameters are independent of the mode of data transfer. ASCII and RTU are supported over stream sockets and the Modbus maps are independent of the transport mechanism.

2.2 Modbus TCP

This protocol is based on the Modbus/TCP specification. The connection mechanism is TCP/IP which secures data transfer between any machines connected on a network. The Modbus protocol is encapsulated into the TCP/IP frame.

The data transfer within the Modbus part of the protocol is standard Gould/Modicon Modbus. The only significant difference is that the Modbus CRC is **not** used: the CRC with the TCP/IP protocol is used **instead**.

Important

Ports 6000, 6001, 6002, 6010, 6011, and 6012 are **reserved** for clients using the FloBoss S600+ specific Modbus function code 65 and should **not** be used.

2.3 Serial

In serial mode the data links communicate over standard serial lines with these specifications:

- **Baud rates:** 2400, 4800, 9600, 19200, 38400, and 57600
- **Data bits**: 7 (ASCII, required for Modbus ASCII) and 8 (RTU, required for Modbus RTU).
- **Stop bits**: 1, 2.
- **Parity**: None, Even, Odd

2.4 Types and Capabilities of Serial Ports

The FloBoss S600+ supports the following serial ports:

- **Com. Port 1:** Reserved for the front panel display and keypad.
- **Com. Port 2:** Reserved for the Config600™ Configuration transfer tool.
- **Com. Ports 3 and Port 4:** Dedicated RS-232 with RTS-CTS handshaking capabilities
- **Com. Ports 5, 6, and 7:** RS-485/RS-422 capability (selectable using jumper links on the P152 CPU board). RTS line is controlled automatically by the P152 module's UART and rapid turnaround is possible on the 485 links.
- **Com. Port 8:** Reserved for I/O board communications.
- **Com. Ports 9, 10, 11, and 12:** Dedicated RS-485.

Section 3. Network Interface

The FloBoss S600+ includes two independent 10 or 100 MB/s, full-duplex Ethernet interfaces. The system firmware (binary.app 06.30a and later) provides a full TCP/IP stack implementation on both interfaces.

Standard supported features include:

- NTP
- Embedded Webserver
- Socket-based Modbus communications

Note

As per standard cybersecurity practices, the FloBoss S600+ should reside on a closed Local Area Network (LAN). If this is not possible, additional port blocking **must** be performed to ensure the FloBoss S600+ has no route to the Internet.

It is recommended that the 2 network ports are on separate subnets.

3.1 Subnet Masks

The subnet mask can be used to further control the access to the network ports.

For example:

Network Port 1

- IP Address = 192.168.102.2
- Subnet = 255.255.255.128
- Gateway = 192.168.102.127

These settings allow connection by devices in the range 192.168.10.1 to 192.168.102.126.

3.2 Network Cable Unplugged

By default, if a network cable is unplugged from an FloBoss S600+ network port, the FloBoss S600+ stops responding to messages directed to the IP address of that network port.

Within the cold start menu of the FloBoss S600+ is an option to disable this functionality (Located under NETWORK SETUP / WIRE UNPLUG GFC) meaning that if a network cable is unplugged, the FloBoss S600+ then routes all network traffic through the remaining connected port. However, it is recommended that the smart switches within the network perform this re-routing if required.

3.3 Network Configuration (06.30 and earlier)

Prior to binary.app 06.30a, only one gateway could be entered. This was entered for network port 1 but the gateway would be used by the network port for which it was part of the subnet.

In the example below, messages to an address outside of the 192.168.102.x range would be sent to gateway 192.168.102.1 via network port 1 (the gateway is on network port 1 subnet).

Network Port 1

- IP Address = 192.168.102.2
- Subnet = 255.255.255.128

- Gateway = 192.168.102.1

Network Port 2

- IP Address = 192.168.102.130
- Subnet = 255.255.255.128

In the second example below, messages to an address outside of the 192.168.102.x range would be sent to gateway 192.168.102.129 via network port 2 (the gateway is on network port 2 subnet).

Network Port 1

- IP Address = 192.168.102.2
- Subnet = 255.255.255.128
- Gateway = 192.168.102.129

Network Port 2

- IP Address = 192.168.102.130
- Subnet = 255.255.255.128

3.4 Network Configuration (06.30a onwards)

From binary.app 06.30a onwards, both network ports had a full TCP/IP stack implementation. This allows separate gateways to be configured for each network port if required.

Section 4. Modbus

Modbus is the standard interface to the FloBoss S600+ The protocol is based on Gould/Modicon Modbus. The FloBoss S600+ supports both Modbus slave and Modbus master interfaces.

The following Config600 tools (Config Transfer, Remote Archive Uploader and the Logicalc Editor connection) interface to the FloBoss S600+ using Modbus. This link uses the special "function code 65," with specially defined FloBoss S600+ sub-functions to provide file transfer, system edit commands, and other specialized functionality.

4.1 Configuration

The system firmware supports up to **20** communications links. Links can have individual Modbus maps or can share common Modbus maps depending on the configuration requirements.

Configuration has two parts:

- **Link Configuration**
  This step defines the data link's port or socket, its master or slave status, and whether it uses RTU, ASCII, or Modbus TCP. You enter this information using Config600 when the link is initially assigned, but the link configuration can be modified later if required.
- **Map Configuration**
  This step assigns database points and fields to Modbus coils, inputs and registers using a text file created using Config600's Modbus Editor. Changes to the Modbus map must be performed offline and the modified configuration downloaded to the FloBoss S600+ for the changes to take effect.

4.2 Master

In master mode, the device communicates with up to 10 slaves on a single-drop link. You can configure several masters if necessary.

4.3 Slave

In slave mode, the device waits for polls from an external master and responds to polls when they are requested. You can define several slaves if necessary. The slave address is configurable per link, so a single FloBoss S600+ can have several slave addresses if required.

Section 5. RTU/ASCII Protocol

The FloBoss S600+ fully supports both RTU Modbus and ACSII Modbus. A single configuration switch enables you to select one or the other. Other communication parameters exist independently of the ASCII/RTU switch.

- **RTU Mode**
  In RTU mode, you must configure the link for 8 data bits. No message header or trailers are included. The checksum is the 16-bit CRC specified in the Modbus specification.
- **ASCII Mode**
  In ASCII mode, you would normally configure the link for 7 data bits, although the FloBoss S600+ also supports 8 data bits. The message starts with the ASCII Modbus start character (the colon, or ":"). The checksum is the 8-bit LRC defined in the

Modbus specification. The message terminates with the ASCII Modbus trailer characters CR followed by LF.

Section 6. Implemented Modbus Functions

Implemented Modbus functions include:

- **Function 1:** Read Output Status
- **Function 2:** Read Input Status
- **Function 3:** Read Output Registers
- **Function 4:** Read Input Registers
- **Function 5:** Write Single Coil
- **Function 6:** Write Single Register
- **Function 8:** Loopback
- **Function 15:** Write Multiple Coils
- **Function 16:** Write Multiple Registers

Each of these functions is described in this section.

Notes

- These message formats **exclude** the message header (ASCII only), the CRC/LRC, and message trailer (ASCII only) since these are specific to the data transfer mode.
- The Gould/Modicon Modbus protocol specifies the address range for each function code. However. the FloBoss S600+ Modbus configuration is flexible in that it does not have to adhere to these address ranges although they are stated below for clarity.

6.1 Function Code 1: Read Output Status

Function code 01 reads the output coils (Gould/Modicon addresses 00001–09999), reading up to 2040 coils per poll in non-extended mode. Refer to *Section 7, Message Length Mode* for more information on non-extended and extended mode.

Poll Format	
Address	1 byte
Function	1 byte

Poll Format	
Start Coil	2 bytes
Num Coils	2 bytes
Response Format	
Address	1 byte
Function	1 byte
Byte Count	1 or 2 bytes **Note**: Byte count may be 8-bit or 16-bit
Data byte 1	1 byte (coils 0–7)
Data byte 2	1 byte (coils8–15)
…	
Data byte N	1 byte (coils)

6.2 Function Code 2: Read Input Status

Function code 02 reads the input coils (Gould/Modicon addresses 10001–19999), reading up to 2040 coils per poll in non-extended mode.

Poll Format	
Address	1 byte
Function	1 byte
Start Input	2 bytes
Num Inputs	2 bytes
Response Format	
Address	1 byte
Function	1 byte
Byte Count	1 or 2 bytes **Note**: Byte count may be 8-bit or 16-bit
Data byte 1	1 byte (inputs 0–7)
Data byte 2	1 byte (inputs 8–15)
…	
Data byte N	1 byte (inputs…)

6.3 Function Code 3: Read Output Registers

Function code 03 reads the output registers (Gould/Modicon addresses 40001–49999); data can be configured at every address.

Poll Format	
Address	1 byte
Function	1 byte
Start Item	2 bytes
Num Items	2 bytes
Response Format	
Address	1 byte
Function	1 byte
Length	1 or 2 bytes **Note**: Length can be 8-bit byte count, 8-bit item count; 16-bit byte count, or 16-bit item count.
Data item 1	2, 4, or 8 bytes
Data item 2	2, 4, or 8 bytes
…	
Data item N	2, 4, or 8 bytes

6.4 Function Code 4: Read Input Registers

Function code 04 reads the input registers (Gould/Modicon addresses 30001–39999); data can be configured at every address.

Poll Format	
Address	1 byte
Function	1 byte
Start Item	2 bytes
Num Items	2 bytes
Response Format	
Address	1 byte
Function	1 byte
Length	1 or 2 bytes **Note**: Length can be 8-bit byte count, 8-bit item count, 16-bit byte count, or 16-bit item count.
Data item 1	2, 4, or 8 bytes
Data item 2	2, 4, or 8 bytes

Poll Format	
...	
Data item N	2, 4, or 8 bytes

6.5 Function Code 5: Write Single Coil

Function code 05 writes a single coil, either on (1) or off (0), to any valid coil address (Gould/Modicon addresses 00001–09999).

Poll Format	
Address	1 byte
Function	1 byte
Coil Number	2 bytes
Coil Value	2 bytes
Response Format	
Address	1 byte
Function	1 byte
Coil Number	2 bytes
Coil Value	2 bytes

6.6 Function Code 6: Write Single Register

Function code 06 writes a value to a single holding register in standard Modbus format to any valid holding register address (Gould/Modicon addresses 40001–49999).

Poll Format	
Address	1 byte
Function	1 byte
Item Number	2 bytes
Value	2 bytes
Response Format	
Address	1 byte
Function	1 byte
Item Number	2 bytes
Value	2 bytes

Note

The write single register function applies **only** to simple 16-bit registers. The float and double formats are **not** supported.

6.7 Function Code 8: Loopback

Function code 08 performs a loopback test. If successful a loopback test returns the poll received. There are no variable parameters available on this function code.

Poll Format	
Address	1 byte
Function	1 byte
Value 1	2 bytes
Value 2	2 bytes
Response Format	
Address	1 byte
Function	1 byte
Value 1	2 bytes
Value 2	2 bytes

6.8 Function Code 15: Write Multiple Coils

Function code 15 writes multiple coils on (1) or off (0). Enter data as a string of up to 8 digits (0s and 1s). The RHS entry represents the LSB. Data is written to any valid coil address (Gould/Modicon addresses 00001–09999).

Poll Format	
Address	1 byte
Function	1 byte
Start Coil	2 bytes
Num Coils	2 bytes
Byte Count	1 or 2 bytes **Note**: Byte count may be 8-bit byte count or 16-bit byte count.
Data byte 1	1 byte (coils 0–7)
Data byte 2	1 byte (coils 8–15)
…	
Data byte N	1 byte (coils)

Poll Format	
Response Format	
Address	1 byte
Function	1 byte
Coil Number	2 bytes
Coil Value	2 bytes

6.9 Function Code 16: Write Multiple Registers

Function code 16 writes a value to a multiple holding register in standard Modbus format to any valid holding register address (Gould/Modicon addresses 40001–49999).

Poll Format	
Address	1 byte
Function	1 byte
Start Item	2 bytes
Num Items	2 bytes
Byte Count	1 or 2 bytes **Note**: Byte count may be 8-bit byte count or 16-bit byte count.
Data item 1	2, 4, or 8 bytes
Data item 2	2, 4, or 8 bytes
…	
Data item N	2, 4, or 8 bytes
Response Format	
Address	1 byte
Function	1 byte
Item Number	2 bytes
Num Items	2 bytes

Section 7. Message Length Mode

Using standard Gould/Modicon Modbus, all variable length messages include an 8-bit byte count field which allows the receiver to determine the amount of data in the message. In some applications, the use of this field is not the actual byte count but the number of data items in the message. This variation only applies to *register* messages. This is also referred to as non-extended mode.

Emerson has provided the option to make this field a 16-bit value, which simply allows more data to be transferred in each message. This is also referred to as “extended mode.”

Note

This functionality **does not** adhere to strict Gould/Modicon Modbus functionality.

Available message length modes are:

- **8-bit byte count**

  The byte count field consists of a single byte, which determines the number of data bytes in the message. For example, with data item length 2 bytes:

Address	Function	8-bit byte count	Data	Checksum
01	03	02	00 01	xx

- **8-bit item count**

  The item count field consists of a single byte, which determines the number of data items in the message. For example, with data item length 2 bytes:

Address	Function	8-bit item count	Data	Checksum
01	03	02	00 01 00 02	xx

- **16-bit byte count**

  The byte count field consists of two bytes, which determine the number of data bytes in the message. For example, with data item length 2 bytes:

Address	Function	16-bit byte count	Data	Checksum
01	03	01 00	00 01 00 02 … up to 256 bytes	xx

- **16-bit item count**

  The item count field consists of two bytes, which determine the number of data items in the message. For example, with data item length 2 bytes:

Address	Function	16-bit item count	Data	Checksum
01	03	01 00	00 01 00 02 ... up to 256 items	xx

Section 8. Register Formats

The *Modicon Modbus Protocol Reference Guide* (publication PI-MBUS-300, Rev. J, from Modicon, Inc.) does not specify how numbers greater than 16 bits (that is, 32-bit integers and single- and double-precision real numbers) should be transferred.

The Modbus protocol has 16-bit elements ("*registers*") that are only suitable for transferring 16-bit integer data. Since no standard exists, Emerson has developed the following formats to transfer numerical data to and from third-party systems.

- Float
- Rosemount
- Double
- Phillips Float
- Phillips Double
- Enron 16-bit
- Enron 32-bit
- Enron Float
- Scale 0 nnnn

Each of these formats is described in this section.

8.1 Single-Precision Floating Point: MSB First

[FORMAT FLOAT]
[FORMAT ENRON FLOAT]
[FORMAT PHILLIPS FLOAT]

Formats numerical items as IEEE format single-precision 32-bit floating point numbers, with a standard "MSB first" byte order.

Four (4) bytes are used to represent the item, with the bytes being distributed across register addresses according to the Addr/Item selection (A = MSB, etc.):

Addr/Item = 1

- Address 1 = ABCD (Item 1)
- Address 2 = ABCD (Item 2)

Addr/Item = 2

- Address 1 = AB (Item 1)
- Address 2 = CD (Item 1)

Addr/Item = 4

- Address 1 = A (Item 1)
- Address 2 = B (Item 1)
- Address 3 = C (Item 1)
- Address 4 = D (Item 1)

8.2 Single-Precision Floating Point: Word Swap

[FORMAT ROSEMOUNT]

Formats numerical items as IEEE format single-precision 32-bit floating point numbers, with a “word swap” byte order.

Four (4) bytes are used to represent the item, with the bytes being distributed across register addresses according to the Addr/Item selection (A = MSB, etc.):

Addr/Item = 1

- Address 1 = CDAB (Item 1)
- Address 2 = CDAB (Item 2)

Addr/Item = 2

- Address 1 = CD (Item 1)
- Address 2 = AB (Item 1)

Addr/Item = 4

- Address 1 = C (Item 1)
- Address 2 = D (Item 1)
- Address 3 = A (Item 1)
- Address 4 = B (Item 1)

8.3 Double-Precision Floating Point: MSB First

[FORMAT DOUBLE]
[FORMAT PHILLIPS DOUBLE]

Formats numerical items as IEEE format double-precision 64-bit floating point numbers, with a standard ”MSB first” byte order.

Eight (8) bytes are used to represent the item, with the bytes being distributed across register addresses according to the Addr/Item selection (A = MSB, etc.):

Addr/Item = 1

- Address 1 = ABCDEFGH (Item 1)
- Address 2 = ABCDEFGH (Item 2)

Addr/Item = 2

- Address 1 = ABCD (Item 1)
- Address 2 = EFGH (Item 1)

Addr/Item = 4

- Address 1 = AB (Item 1)
- Address 2 = CD (Item 1)
- Address 3 = EF (Item 1)
- Address 4 = GH (Item 1)

8.4 16-bit Integer

[FORMAT ENRON-16-BIT]

Formats numerical items as 16-bit SIGNED integers, ranging from –32767 (hex 8001) to 32767 (hex 7FFF).

Two (2) bytes are used to represent the item, with Addr/Item = 1 the only permissible option (A = MSB, etc.):

Addr/Item = 1

- Address 1 = AB (Item 1)
- Address 2 = AB (Item 2)

8.5 32-bit Integer

[FORMAT ENRON 32-BIT]

Formats numerical items as 32-bit SIGNED integers, ranging from -2,147,483,647 (hex 80000001) to 2,147,483,647 (hex 7FFFFFFF).

Four (4) bytes are used to represent the item, with the bytes being distributed across register addresses according to the Addr/Item selection (A = MSB, etc.):

Addr/Item = 1

- Address 1 = ABCD (Item 1)
- Address 2 = ABCD (Item 2)

Addr/Item = 2

- Address 1 = AB (Item 1)

- Address 2 = CD (Item 1)

8.6 Scaled Integers

[FORMAT SCALE 0-999]
[FORMAT SCALE 0-4096]
[FORMAT SCALE 0-9999]

Formats numerical items as 16-bit integers, scaled between 0 and 999/4096/9999.

Two (2) bytes are used to represent the item, with Addr/Item = 1 the only permissible option (A = MSB, etc.):

Addr/Item = 1

- Address 1 = AB (Item 1)
- Address 2 = AB (Item 2)

Only the object types in the following table can be used with the Scaled Integer option. The table also shows which object fields/fixed values are used to scale the object:

Object Types	Native Lo	Native Hi
ADC, DAC	LOSCALE	HISCALE
PRT, CALCALMITEM, DPCELL	LOLOLIM	HIHILIM
KPREALARR	0	100
KPREAL	0	1000

When a Scaled Integer is read from the flow computer, the value on Modbus map is calculated using:

$$Scaled\ Integer\ Value = Trunc\left[\frac{(Native\ Value - Native\ Lo) \times Scale\ Full}{(Native\ Hi - Native\ Lo)}\right]$$

where:

- Native Value = Native value of object field as it appears in flow computer database
- Native Lo = Low range value for native object, from table above
- Scale Full = 999/4096/9999, depending on Modbus format chosen
- Native Hi = High range value for native object, from table above

For example, a flow rate (CALCALMITEM) of 300 m^3/h has a Native Lo (LOLOLIM) of 0 m^3/h and a Native Hi (HIHILIM) of 1000 m^3/h, using a format of SCALE 0-9999:

$$Scaled\ Integer\ Value = Trunc\left[\frac{(300 - 0) \times 9999}{(1000 - 0)}\right]$$

$$Scaled\ Integer\ Value = Trunc[2999.7]$$

$$Scaled\ Integer\ Value = 2999$$

When a Scaled Integer is written to the flow computer, the native value is calculated using:

$$Native\ Value = \left[\frac{(Native\ Hi - Native\ Lo) \times Scaled\ Integer\ Value}{Scale\ Full}\right] + Native\ Lo$$

Section 9. Modbus Configuration Files

Whenever a Modbus communication task starts, it is associated with a Modbus configuration file (mbxxxx.txt). This file defines the map that links Modbus addresses to FloBoss S600+ database points.

All configuration files are ASCII text files which may be created manually if required. However, Config600 software program includes Modbus configuration file generators for most common applications. Always use these file generators. You can then edit the generated files using either the Config600 Modbus editor or a text editor (such as Notepad). Refer to the *Chapter 14* of the *Config600™ Configuration Software User Manual* (part D301220X412) for more information on the Modbus editor.

There are two basic file types, master and slave. Both share many common features since the slave configuration file is a subset of the master configuration file. The # symbol is used to indicate a comment. When a line is read, any characters following a # symbol are ignored. The overall layout of the configuration files depends on whether the file is for master or slave use:

Format of Master Configuration File

Title
    Header Section
    Master Section
Slave 1 Data
Slave 2 Data
...
Slave N Data

Format of Slave Configuration File:

Title
Header Section
Slave 1 Data

9.1 Config File Title

This is simply a comment at the start of the file describing its function, version history, etc. The FloBoss S600+ ignores this comment.

9.1.1 Config File Header

This block, used by both master and slave links, is mandatory. Four commands are supported, all of which are enclosed in brackets ([]):

- **[TX BUFF SIZE <NNNN>]**
- **[RX BUFF SIZE <NNNN>]**

  These lines determine the size in bytes of the message transmit (TX) and receive (RX) buffers. For normal Modbus operation set them both to **1024** which is the maximum message size serial Modbus supports.

- **[MSG LENGTH MODE <mode>]**

  This field determines how the message length (Byte Count) of the Modbus message is handled (refer to *Section 7, Message Length Mode* for more information). The basic Modbus specification defines this field as a single byte which defines the number of bytes of data in the message. However, many implementations change this field to be an item count, such as *the number of data items in the message*.

  Additionally, to increase the amount of data which may be transferred, the FloBoss S600+ supports a 16-bit value for this field. This is **not** compatible with standard Modbus and is used **only** when communicating with compatible systems, such as the legacy Daniel DMS supervisory. It should **not** be used for any other third-party devices.

  The available values for <mode> are:

  - BYTE_8
  - BYTE_16
  - ITEM_8
  - ITEM_16

- **[CHECKSUM <status>]**

  This field is used to enable or disable checksum checking on received data. It is intended for test purposes only. The available values for <status> are.

  - TRUE
  - FALSE

Example Header:

[TX BUFF SIZE 256]
[RX BUFF SIZE 256]
[MSG LENGTH MODE BYTE_8]
[CHECKSUM TRUE]

9.2 Config File Master Section

This section is required **only** for the master task. It defines the master's poll loop settings, the list of slave machines with which the master communicates, and a series of polls to perform. Six commands are supported, all of which are enclosed by brackets ([]):

- **[POLL DELAY <NNN>]**

  This specifies the delay between messages (polls). The value is in system ticks (a tick is 1/60 of a second)

- **[LOOP DELAY <NNN>]**

  This specifies the delay at the end of the poll loop (i.e. the time between the last message and starting the first message again). The value is in system ticks (a tick is 1/60 of a second).

- **[RETRY LIMIT <NNN>]**

  This specifies the number of retries the master should perform for a single message before classing it as failed. The retry counter is incremented following a timeout (see below).

- **[TIMEOUT <NNN>]**

  This specifies the amount of time (in seconds) that the master waits for a valid response from the slave before it increments the retry counter. During this timeout period, the master attempts to poll the slave every 3 seconds. For example, for a TIMEOUT period of 60, the master attempts to poll 20 times.

- **[SLAVES <A> <B> <C> <D>]**

  This command specifies a list of up to 10 slave addresses with which the master task communicates.

- **[SLAVE STATUS OBJ <KPINTARR Number> <KPINTARR Field>]**

  This command specifies the KPINTARR and Field (stream number) used to indicate the status of the link.

  For each configured stream, the Modbus master task updates the KPINTARR with:

Value	Description
0	Link OK

Value	Description
1	Link disabled
2	Telemetry failed

- **[SLAVE CONTROL OBJ <KPINTARR Number>]**

  This command specifies the KPINTARR used to indicate if the link is enabled or disabled. The first field of the KPINTARR refers to the first stream that uses the link, the second field of the KPINTARR refers to the second stream that uses the link, and so on.

Value	Description
0	Link disabled
1	Link enabled

- **[SLAVE ADDRESS OBJ <KPINTARR Number>]**

  This command specifies the KPINTARR used to set the slave address that the link will poll for.

  The first field of the KPINTARR refers to the first stream that uses the link, the second field of the KPINTARR refers to the second stream that uses the link, and so on.

Notes

- For Station Gas Chromatographs, separate KPINTARR's are used for Chromat A and Chromat B so only the **first** field of each KPINTARR is used.
- If this object is configured, the addresses configured in the KPINTARR take precedence over the values configured in the **[SLAVES <A> <B> <C> <D>], [MESSAGE ADDR:<addr> FUNC:<func> START:<start> NUM:<num> TRIGGER:KPINT <n>]** and **[SLAVE ADDRESS <A>]** settings.

- **[MESSAGE ADDR:<addr> FUNC:<func> START:<start> NUM:<num> TRIGGER:KPINT <n>]**

  The message command defines a single message which the master task performs. Any number of messages may be included. The master task performs the messages according to their order in the file.

  - <addr> is the Modbus slave address (0...255).
  - <func> is the Modbus function code for the message.
  - <start> is the data start address for the message.
  - <num> is the number of addresses in the message.
  - <n> is the database index for the KPINT trigger object used to control the poll.

The trigger field is optional.

- If it is **not** present, the poll is always performed.
- If it **is** present, a value of 1 triggers the poll.
- Once all processing of the message has occurred, the trigger value resets to 0.

From a timing perspective, the poll delay for any triggered messages is included in the overall loop time (even if the trigger value is **not** set to 1, that is, not triggered). When the trigger value is set to 1, after one complete loop period, the triggered message is sent. Then, after one poll delay period, non-triggered messages continue.

Example Master Section

```
# MASTER SETTINGS
[POLL DELAY 10]
[LOOP DELAY 10]
[RETRY LIMIT 3]
[TIMEOUT 5]
# LIST OF SLAVE ADDRESSES
[SLAVES 3]
# SLAVE STATUS OBJECT COR SLAVE STATUS
[SLAVE STATUS OBJ 29 0]
# SLAVE CONTROL OBJECT COR SLAVE ENABLE
[SLAVE CONTROL OBJ 38]
# SLAVE ADDRESS OBJECT COR SLAVE ADDR
[SLAVE ADDRESS OBJ 46]
# MAIN POLL LOOP - PERFORMED CONSTANTLY
[MESSAGE ADDR:3 FUNC:3 START:0 NUM:1]
[MESSAGE ADDR:3 FUNC:3 START:124 NUM:2]
[MESSAGE ADDR:3 FUNC:3 START:246 NUM:20]
# TRIGGER FOR RESET
[MESSAGE ADDR:3 FUNC:5 START:2 NUM:1 TRIGGER:KPINT 357]
```

9.3 Config File Slave N Section

Each slave to which the master communicates requires a slave data block. The block starts with the command:

- **[SLAVE ADDRESS <N>]**

  <N> can be any valid Modbus slave address value (from 1 to 255).

Following this statement is the data map for the slave.

9.4 Config File Data

This section applies to both master and slave links. It defines the relationship between Modbus addresses and FloBoss S600+ database points.

The slave has a single map section, while the master can have up to 10 slave sections, each of which is preceded by the [SLAVE ADDRESS N] command. The map is sub-divided into coil, input, and register sections, each of which forms independent blocks of data. Within each of these blocks there may be any number of further sub-blocks.

The map section has these major blocks:

- **[SECTION COILS]**
  Indicate the start of the Modbus coil data.
- **[SECTION INPUTS]**
  Indicate the start of the Modbus input data.
- **[SECTION REGISTERS]**
  Indicate the start of the Modbus register data.

Note
The Gould/Modicon Modbus protocol specifies the address range for each function code. However, the FloBoss S600+ Modbus configuration is flexible in that it does not have to adhere to these address ranges. Therefore, the BASE ADDRESS specified is the actual address that is used (that is, the Gould/Modicon offset is **not** added to the base address).

9.5 Config File Data Coil

The coil section can contain any number of blocks of coil data. Each block starts with the command:

- **[BASE ADDRESS <NNNN>]**

  This command determines the base address of the block of data. <NNNN> must be in the range 0–65535. The base address command is followed by a list of data points each of which occupies successive coil addresses.

9.6 Config File Data Input

The input section can contain any number of blocks of input data. Each block starts with the command:

- **[BASE ADDRESS <NNNN>]**

This command determines the base address of the block of data. <NNNN> must be in the range 0–65535. The base address command is followed by a list of data points, each of which occupies successive input addresses.

9.7 Config File Data Registers

The register section can contain any number of blocks of register data. Three commands control how the system packs data into registers:

- **[BASE ADDRESS <NNNN>]**

  This command determines the base address of the block of data. It is followed by a list of data points each of which occupies successive registers. <NNNN> must be in the range 0–65535.

- **[FORMAT <format>]**

  Each block of register data can represent numeric data in a different format.

  Register addressing is probably the most confusing aspect of configuring a Modbus link. The original Modbus specification defines a register as a 16-bit integer value. Strictly speaking, this is the only supported register data format. However, most systems split larger formats over several registers so that a single data item occupies 1, 2, or 4 registers (respectively 2, 4, or 8 bytes). (refer to *Section 8, Register Formats*, for details of this and the supported formats). Using this mechanism, you can pack longer integers, floats, or doubles into the Modbus message.

- **[ADDRESSES PER ITEM <N>]**

  This command determines the number of Modbus addresses which each data item corresponds to. For 16-bit (2 byte) data it is **always** 1. For 32-bit (4 byte) and 64-bit (8 byte) data it controls the way Modbus register addressing is performed. It is closely related to the FORMAT command (or more accurately the number of bytes per data item) and the number of addresses field contained in the poll.

  For a *slave* link, "the poll" refers to the Modbus request sent by the master. For a *master* link, "the poll" refers to the Message command contained in the Master section.

  If the data is **32-bit**, the number of addresses per item must be set to either **1** or **2**. If the value is set to **1**, the number of addresses contained in the poll is the number of data items (in this case, the number of items and number of registers is the same). If the value is set to **2**, the number of addresses contained in the poll is **2x** the number of data items.

If the data is **64-bit**, the number of addresses per item must be set to either **1** or **4**. If the value is set to **1**, the number of addresses contained in the poll is the number of data items (in this case, the number of items and number of registers is the same). If the number of addresses per item is **4**, the number of addresses contained in the poll is **4x** the number of data items.

ADDRESSES PER ITEM may be set to **–1** for some special data formats. This instructs the Modbus message building code to ignore the number of registers requested. This setting is typically used for Enron report data (where the number of registers is used to define the report number, but the number of registers to be transmitted is determined by the report layout). Refer to the Enron Corp document *Specifications and requirements for an Electronic Flow Measurement Remote Telemetry Unit* (Revision dated December 5th, 1994), for more details of this functionality.

Combinations of FORMAT and ADDRESSES PER ITEM commands should support the most commonly used register formats. The ADDRESSES PER ITEM command must be followed by a list of data points, each of which must occupy successive register addresses.

9.8 Discrete Data Point Format

Discrete data points appear in the Coils or Inputs section. All the data that makes up a block is defined using a series of consecutive lines in the configuration file which refer to objects in the FloBoss S600+ database. Each line represents a point in the database. You can add any number of lines to a block. Each block refers to successive coils or inputs.

The format of the line is:

<OBJECT_TYPE> <OBJECT_INDEX> <OBJECT_FIELD> < OBJECT_BITNO>

where:

<OBJECT_TYPE>, <OBJECT_INDEX>, and <OBJECT_FIELD> define a unique point in the database using the object identification format used by the reports, display, and communication configuration files. <OBJECT_BITNO> is the bit within the specified database point which is mapped to the coil or input, such as:

ADC 3 MEASURED 0

Refer to the auto-generated slave configuration file (mbslave.txt) for many examples of the format used.

9.9 Numeric Data Point Format

Numeric data points appear in the Register section. The data which makes up a block is defined using a series of consecutive lines in the configuration file which refer to objects

in the FloBoss S600+ database. Each line represents a point in the database. You can add any number of lines to a block. Each block refers to successive registers.

The format of the line is:

<OBJECT_TYPE> <OBJECT_INDEX> <OBJECT_FIELD>

where:

<OBJECT_TYPE>, <OBJECT_INDEX>, and <OBJECT_FIELD> define a unique point in the database using the object identification format used by the reports, display, and communication configuration files, such as:

CALCALMITEM 26 INUSE # [MASS-FR)

Refer to the auto-generated slave configuration file (mbslave.txt) for many examples of the format used.

For customer service and technical support, visit *Emerson.com/Guardian*.

North America and Latin America:
Emerson Energy and Transportation Solutions
6005 Rogerdale Road
Houston, TX 77072 U.S.A.
T +1 281 879 2699 | F +1 281 988 4445
Emerson.com/SCADAforEnergy

United Kingdom:
Emerson Process Management Limited
Fosse House, 6 Smith Way
Grover Park, Enderby
Leicester LE19 1SX UK
T +44 0 870 240 1987

Europe:
Emerson S.R.L
Regulatory Compliance Shared Services Department
Company No. J12/88/2006
Emerson 4 Street
Parcul Industrial Tetarom 11
Romania
T +40 374 132 000

Middle East/Africa:
Emerson Energy and Transportation Solutions
Emerson FZE
P.O. Box 17033
Jebel Ali Free Zone – South 2
Dubai U.A.E.
T +971 4 8118100 | F +971 4 8865465

Asia-Pacific:
Emerson Energy and Transportation Solutions
1 Pandan Crescent
Singapore 128461
T +65 6777 8211| F +65 6777 0947