FMC **Smith Meter Inc.**
An FMC Corporation subsidiary

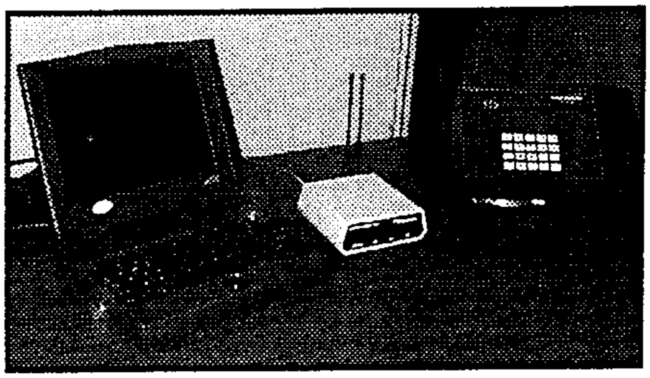
Electronic Crude Oil Gathering System

TCP-CO

Specifications

Bulletin SS0625P

The *Smith Electronic Crude Oil Gathering System* may consist of three components: a TCP cab-mounted flow computer, an optional transceiver for RF communications, and the routing and dispatch data processing software. The heart of the system is the TCP-CO microprocessor-based flow computer, specifically designed for the custody transfer of crude oil at production lease sites. The optional transceiver transmits data to and from the TCP-RAD and the TCP-CO.



Features

- **Improved accuracy by on-board metering**
- **Automatic temperature compensation**
- **Automatic sampling**
- **Automatic BS&W**
- **Automatic Netting of Load Ticket**
- **Computerized routing and dispatch**
- **Instantaneous data recording**
- **Reduced manpower requirements**
- **Improved safety**

Applications

Typical applications include electronic measurement, data recording, and data transfer of crude oil gathering by truck. State-of-the-art meters combined with these flow computing devices eliminate the errors arising from current manual gaging, reduce loading time, and drastically improve safety. The package will include on-board cab-mounted metering electronics, providing online measurements currently obtained by hand gaging. The measurements include volumes (net and gross), oil, online density, automatic sampling,and BS&W monitoring. The data can be transmitted to an onboard printer or to a host computer. This system eliminates errors from current manual gauging methods, reduces loading time, improves safety and automates data transfer.

The flexible system design may also be applied to fixed truck unloading stations. The load lease information is tracked automatically through the RAD system, providing the possibility of unmanned unloading stations with instaneous transfer of data to an off-site central computer system.

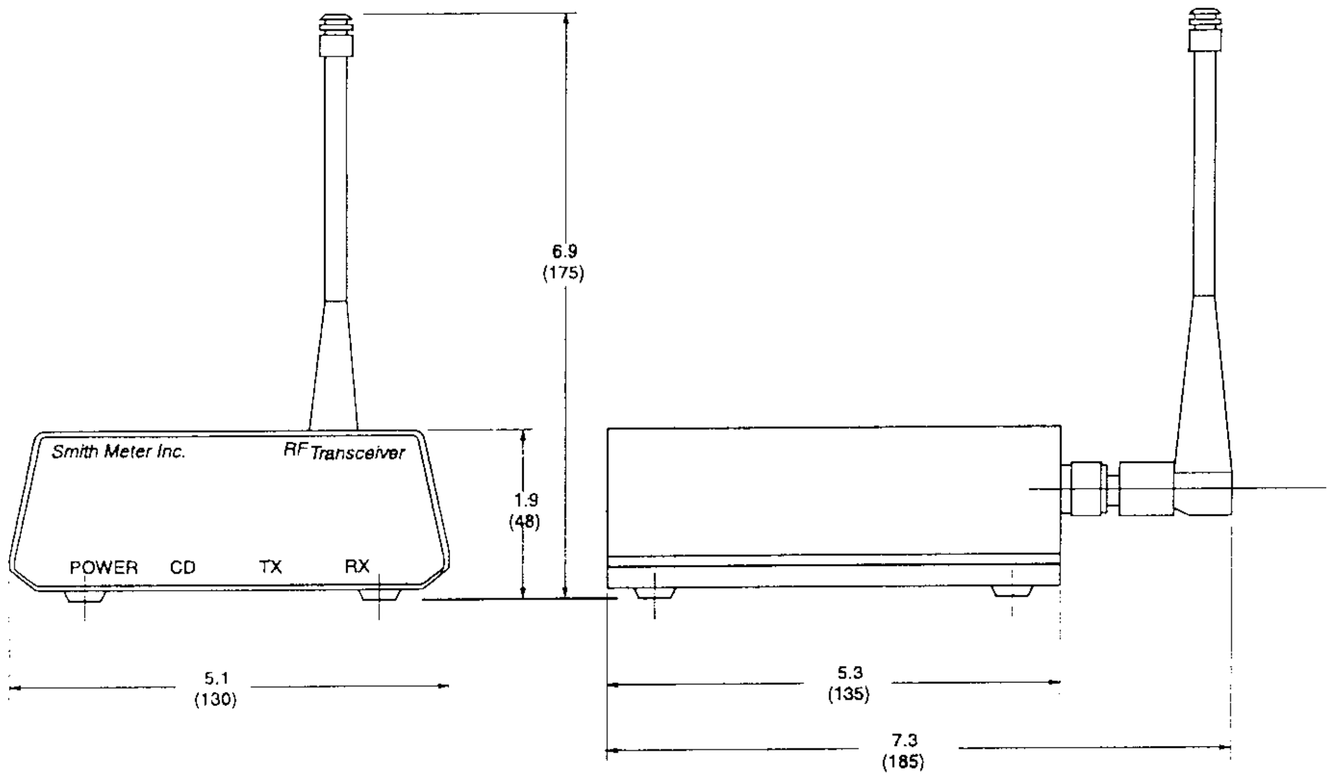
System Operation

The TCP-CO electronic crude oil gathering system is designed to automate the routing, dispatching, and data recording, and improve the accuracy of crude oil gathering. The system includes the electronic equipment, TCP-CO electronic totalizer, TCP-RAD computer program, an optional radio frequency transmitter and receiver, and a meter.

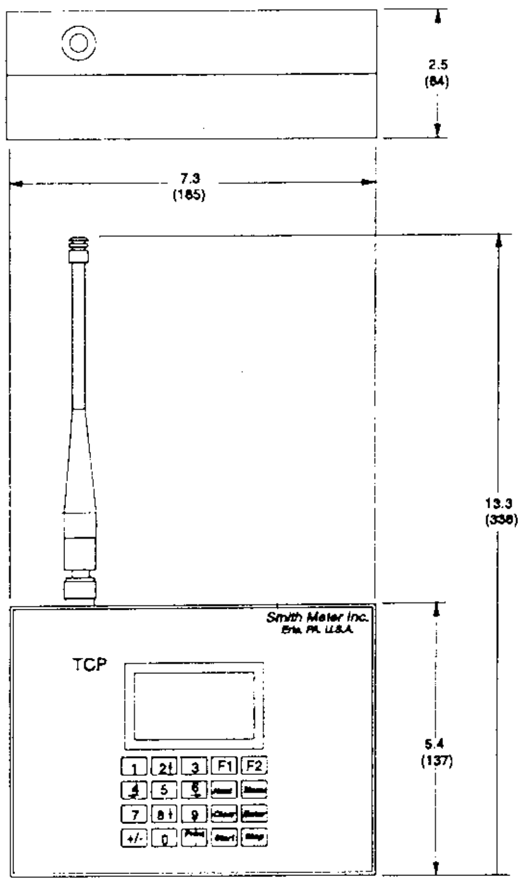
The TCP-CO Electronic Totalizer is designed to be mounted in the cab of a truck. It is wired to the meter, which is mounted on the back of the truck, and either receives pulses from the meter or, in the case of a Smith ST Mass Meter, communicates directly with the meter. The TCP-RAD computer program is installed on a computer in a dispatch office or loaded onto a portable laptop computer. The TCP-RAD and the TCP-CO totalizer will communicate data either via a communications line or through radio frequency communications. The TCP-RAD can be used to program the parameters in the TCP-CO, set up routes and dispatch schedules that are downloaded to the TCP-CO, and gather data from the TCP-CO at the end of a shift or receipt. Once the data has been received from the TCP-CO, the TCP-RAD will store the data in a text file that can be easily transferred to a host system for the purposes of record-keeping, reporting, auditing, etc.

The TCP-RAD is a Windows application designed to streamline and organize the routing and dispatch of tank trucks for crude oil gathering tank collection operations. It is designed to be in operation at the truck terminal, and allow the dispatcher to (1) configure his TCP-CO units, (2) define the scheduled stops for each truck, and (3) retrieve the collected product volume data upon the return of the truck. It will handle the communications interface with the TCP-CO truck-mounted electronics and

Dimensional Diagram



Transceiver



TCP - CO

ON - TRUCK METERING

Application

ude Oil is metered from production storage sites onto trucks to be taken to larger storage sites. The crude oil contains water which can not be metered as oil. The metered volume must be corrected for temperature, density and water content. The oil/water is metered into the truck with a volumetric device, the fluid's temperature is measured at periodic intervals, the water content is determined by centrifuge on periodic samples, and the density is determined by a hydrometer on periodic samples.

At the end of the batch, the average measured temperature, density, and % water content are calculated. The net oil is then determined.

The gross volume, the net volume of oil, the average water content, the average temperature, and the average density are mechanically recorded.

In order to gather samples and to measure the fluid temperature, the driver must get on top of the lease tank. This creates a risk of the driver falling. When the oil is thought to contain H_2S, two people must be present to perform this task in case there is a release and help is needed. The safety risk to personnel and overhead costs have prompted transportation managers to look for alternatives to their current truck loading practices.

he S-Mass Meter (MAY NOW BE KROHNE)

A single coriolis meter can provide accurate mass flow measurement, fluid density and fluid temperature.

Using the meter's mass measurement and its density, **gross volume** can be calculated by the following formula:

$$Q = \text{Mass Rate} \div \text{Density}.$$

Net volume is determined by subtracting the water content and then correcting for temperature based on oil density.

The following benefits should be derived from using a coriolis meter:

(1) A coriolis meter can eliminate the need for the driver getting on the lease tank to take temperature measurements. Thereby, reducing the safety risk and eliminating the overhead associated with the second person in the case of oil containing H_2S.

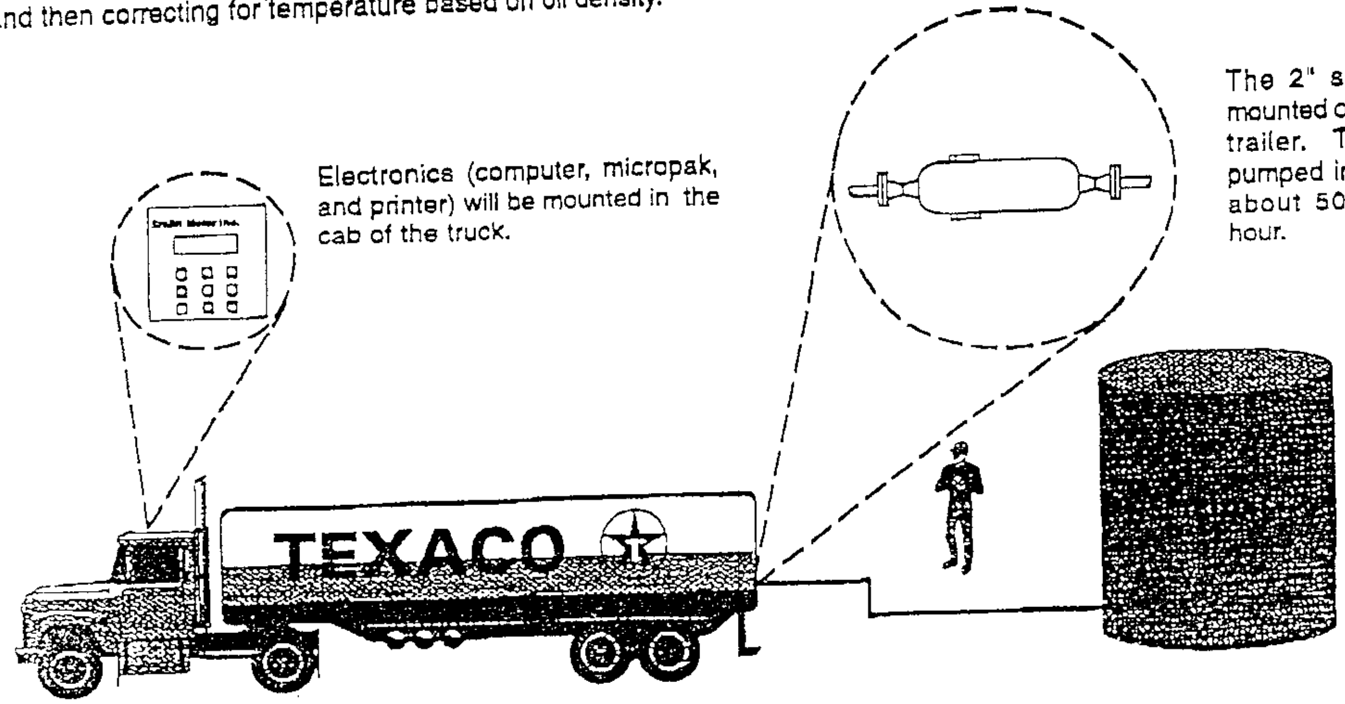
(2) A coriolis meter monitors instantaneous density. This measurement can be used by the meter to detect out of range conditions such as slugs of water or slugs of air. The meter can be configured to ignore its flow measurement under certain conditions.

(3) The meter's ability to measure mass flow allows for the truck to be loaded by weight assuring that the truck is carrying the max allowed load and at the same time the customer can be billed by volume.

(4) A single coriolis meter reduces the instrumentation needed to detect the presence of flow (a flow switch), to provide the measurement of flow, density, and temperature. A coriolis meter reduces initial instrumentation costs, installation costs, and maintenance costs.

System Requirements

- 2" 150# RF flanged Sensor with Micro-Pak
- Flow Computer capable of taking frequency inputs from Micro-Pak and calculating gross volume, net volume of oil, average density, average temperature. The computer should be able to take a live or manual entry for water content.
  The computer should be capable of sending this data to a printer.
- Thermal printer capable of printing a ticket showing the above calculated infomation.


Electronics (computer, micropak, and printer) will be mounted in the cab of the truck.
The 2" sensor will be mounted on its side on the trailer. The oil will be pumped into the truck at about 500 barrels per hour.
TEXACO

FMC Smith Meter Inc.
An FMC Corporation subsidiary

TCP-CO
External Communications Specification

July 21, 1997
Revision 0
Project # RS050197

Prepared by: [illegible]
James M. Pettinato - Associate Software Engineer

Approved by: [illegible]
Robert M. Smith - Project Engineer

Approved by: [illegible]
Constance L. Gibbons - Sr. Software Engineer

Approved by: [illegible]
Dale A. Bohman - Product Marketing Manager

Approved by: [illegible]
David P. Resch - Manager
Electronic Products and Automation

Communications Stack Layers

Hardware Layer

This layer may be manifested by several means. In the TCP application, if there is a hard-wired connection between the TCP-CO and the host, this layer is simply the cable connecting the UARTs of each device. For a radio transceiver, there will be additional hardware required to implement the modulation/demodulation of the RF signal. This layer, in other words, is responsible for converting a serial data packet into a radio signal (IR, etc.) that can be transmitted to and received by another hardware layer device. Therefore the hardware layer could also consist of this alternate hardware that will be used, in conjunction with the packet protocol layer, for RF communications.

Protocol Layer (Packet Layer)

In order to efficiently utilize a transport medium such as radiotransmission, where it is assumed there will be high potential for error due to interference or range limitations, the system will provide a robust error-tolerant protocol layer. This layer provides for the insertion of a packetization function which proves useful in such a transport medium in two ways: it minimizes the impact transient interference has on the overall message by limiting the size of required retransmission of data if said data contained an error, and it allows for the inclusion of a repeater transceiver without extensive software interaction. This layer will reside in the driver code resident on the system running the application. The interface (input and output specification) will be designed in such a manner so that (1) the application layer will not need to be aware of the actual protocol in use; and (2) the protocol layer will provide any interaction required with the hardware layer. This functionality will be hidden from the application layer, allowing the packet layer to perform status checking and other functions such as battery monitoring that may be specific to the currently active hardware layer. Any packet protocol may be used in conjunction with any hardware layer-its output will be simply to place characters in a UART transmit buffer. The application, however, must enforce the proper pairing of protocol layer to hardware layer for efficient and sensible operation. *Note: this pairing will initially be the responsibility of the user configuring the TCP-CO; in the future some method of detecting a combination that is undesirable may be added to the application. Note that even undesirable combinations would still function as expected, although not very efficiently.*

Application Layer

The application layer consists of the actual application tasks on the TCP-CO or the host. The unique aspect of this layer is operation will be independent of the transport medium, i.e. radio communications, IR communications, hard-wired, etc. All communications will be achieved via a virtual interface consisting of a fixed set of functions allowing transmitting, receiving, status checking, etc. with the current communications driver. Hence, the application layer need be concerned only with the specific function command to be executed and any arguments it may require, and also must be prepared to handle all possible responses to the command.

Command Language

The external communications interface of the TCP-CO will be defined by a command format that consists of the following functions:

- 'Ping' or request an acknowledge (response should be an ACK)
- Ack
- Request status information (host to TCP-CO only)
- Report Status
- Upload/Download individual program code entries
- Upload/Download complete program code database
- Request Stop X data (response should be a Transmit Stop X data Command)
- Transmit Stop X data (response should be an ACK)
- Delete Stop X data command (response should be an ACK)
- Upload/Download configurable report definitions
- Error Message

For transmissions initiated by the host, it will be up to the application layer on the receiving TCP-CO to be able to determine which type of command has been received and route it to the appropriate application task to handle the command and invoke the command layer to make the appropriate response. For transmissions initiating on the TCP-CO, the host will immediately take control by sending a status request, hence the initial response to any communications initiated by the TCP-CO will be a status request from the host.

Packet Layer (Radio Protocol)

This protocol will be capable of dividing a message up to 65355 bytes in size into packets that will be transmitted individually, and to receive a similarly sized message that has been so divided. The packet protocol will divide the message into sections not to exceed 248 bytes and transmit the sections in order in packets not to exceed 256 bytes. (For RF, the optimal packet size is smaller, so overall packet size will be limited to 40 bytes.) The packet frame will follow ISO 3309 with the following exceptions: (1) A second address byte will *always* be present and will represent the FROM address; its existence will not be dependent on the first (low order) bit of the first address byte as specified in ISO 3309 – § 4.1. (2) The FCS (Frame Checking Sequence) will be extended to 32 bits as allowed in 3309 for future requirements (§3.6 note 1).

Sample Radio Protocol Packet Frame:

STX	TO	FROM	SEQ#	SIZE	DATA	FCS (CRC32)			
0x02	0x01	0x15	0x00	0x05	nnnnn	MSB	.	.	LSB

The To address will always be the address of the unit intended to receive the communication. The From address will always be the address of the unit originating the communication.

The sequence number is based solely on the previous transmitted sequence number for this transmitter and is not dependent on TO address or message position. The sequence number will have the range of 0-255, and will wrap back to 0 when incrementing from 255. The receiver will always expect the next sequence number except when looking for the first packet of a message, where any sequence number is valid except that a repeat of the previous sequence number within 3 seconds of a successful packet reception will be considered a repeat packet. The receiver will acknowledge the repeat packet, but not act upon it in any other way. The receiver will wait for the next expected packet at least 8 seconds before timing out during a multiple packet message.

The size byte for the packet indicates the size of the data following the size byte, not including the CRC32. As mentioned above, the maximum size for this value will be 248, making the maximum packet size 256 (after stripping the STX). Again, for radio or other noise-laden medium, this size will be limited to 32, making the overall packet size limit 40.

The data field will contain a message or some portion of a message to be recreated at the receiving unit.

Each packet will be acknowledged by the receiving unit via a response packet with the identical sequence number, to/from addresses swapped, a size of 0, and no data bytes. This packet acknowledge is unique and differs from the Acknowledge command described in the command table, which acknowledges a complete message. If the hardware layer is error-prone (i.e., radio or IR link), in the situation where no packet acknowledge is received, the transmitter will attempt to resend up to 5 times with a random delay of 0.5-1.5 secs between each attempt. If the packet is not acknowledged after the final try the transmitter will consider the transmission of the entire message containing that packet a failure. Regardless of success or failure, for the next packet, the transmitter will always increment the sequence number. Each packet may contain a portion of a message or an entire message.

Message (Command) Format

The message format used to transmit commands and responses will be as follows:

Sample Message:

COMMAND	SIZE (MSB)	SIZE (LSB)	Command Arguments	Command CRC32
0x01-0x63	0xNN	0xNN	(optional)	(optional)

The command byte will be one of the commands defined in the following section, *Command Language*. The size word will be the size of the data starting with the first character after the size word up to AND INCLUDING the CRC32 if it exists for this message. (Note: this differs from the packet protocol size argument, which does not count the CRC32 as part of the size.) The command arguments will be dependent on the type of command, and such are described in the next section. The CRC32 associated with the message here is unique from the CRC32 of the individual packets, and its presence is dependent on the size of the message... any message with a size > 64 bytes will have a CRC32 appended to the message. This CRC32 will be calculated upon the data beginning with the first character after the size word until the end of the data.

Command Language

Command 1 - Individual database element read

Data to follow size word:	Register ID - 4 bytes
	Byte 1 - data type
	Byte 2 - function
	Byte 3 - subfunction
	Byte 4 - offset

Good response: Command 2 - Individual database write command
Error response: Command 9 - Error Response

Command 2 - Individual database element write

Data to follow size word: Register ID (see cmd 1) then data value of register
Good response: Command 6 - Ack
Error response: Command 9 - Error Message

Note: A Write command issued in response to a Read command should not expect a response.

Command 3 - block read of database values

Data to follow size word: None
Good response: Command 4 - Block Write command
Error response: Command 9 - Error Message

Command 4 - block write of database values

Data to follow size word: Database data as resides in TCP-CO memory
Good response: Command 6 - Ack
Error response: Command 9 - Error Message

Note: No response if this command was issued in response to a Block Read command.

Database will be segregated by read-only status, i.e., all read-only values will be at the end of the physical structure in memory. All writable members will be contiguous, hence a block of data transferred as a BLOB (Binary Large OBject) will allow a complete program definition to be transferred between the PC and the TCP-CO.

Command 5 - 'Ping' (Ack request)

Data to follow size word: None
Response: Acknowledge command

Command 6 - Acknowledge (Ack)

Data to follow size word: None
Response: None.

Command 7 - Status Request

Data to follow size word: None
Response: Command 8 - Status Response

Command 8 - Status Response

Data to follow size word: 20 byte status. Bytes 1-16 represent the status of stops 1-16 respectively, bytes 17-20 are system status bits.
Response: None.

Status bits for Stop #X

Status byte (x) & 1 = 1	Stop Defined at position X
Status byte (x) & 2 = 2	Stop X completed
Status byte (x) & 4 = 4	Stop X Aborted
Status byte (x) & 8 = 8	Stop X offloaded
Status byte (x) & 16 = 16	reserved for future expansion
Status byte (x) & 32 = 32	"
Status byte (x) & 64 = 64	"
Status byte (x) & 128 = 128	"

Status bits for TCP-CO System

Status byte 1 & 1 = 1	Ready for shift start operations
Status byte 1 & 2 = 2	Ready for shift end operations
Status byte 1 & 4 = 4	Program mode change occurred
Status byte 1 & 8 = 8	reserved for future expansion
Status byte 1 & 16 = 16	"
Status byte 1 & 32 = 32	"
Status byte 1 & 64 = 64	"
Status byte 1 & 128 = 128	"

Status byte 2 & 1 = 1	Alarm condition
Status byte 2 & 2 = 2	reserved for future expansion
Status byte 2 & 4 = 4	"
Status byte 2 & 8 = 8	"
Status byte 2 & 16 = 16	"
Status byte 2 & 32 = 32	"
Status byte 2 & 64 = 64	"
Status byte 2 & 128 = 128	"
Status byte 3	reserved for future expansion
Status byte 4	reserved for future expansion

Command 9 - Error Message

Data to follow size word: 1-byte error code
Response: None.

Error Code Values
0 - No error
1 - Invalid command byte
2 - Invalid command parameters
3 - In Program Mode
4 - Busy (flowing, printing, etc.)
5 - Alarm condition
6 - Insufficient security level access for this function
7 - Requested stop not available
8 - Stop position already in use
9 - Operation out of sequence
10 - Option not available
11 - CRC error in received message

Command 10 - Write Message to TCP-CO display (Not in rev 0)

Data to follow size word: string length word (MSB first, include CRC)
Up to 256 character message string

Good Response: Acknowledge command.
Error response: Command 9 - Error Message

Command 11 - Read Configurable Report Definition

Data to follow size word: None
Good response: Command 12 - Write Configurable Report command
Error response: Command 9 - Error Message

Command 12 - Write Configurable Report Definition

Data to follow size word: up to 65535 byte report definition

Good response: Acknowledge command
Error response: Command 9 - Error Message

Commands 13-15 - Reserved for future expansion

Command 16 + X - Read stop record #X

Data to follow size word: None
Good response: Command 32+X - Write stop record #X
Error response: Command 9 - Error Message

This command instructs the receiving unit to transmit stop information for the stop in the route array in position X where x is 0-15.

Command 32 + X - Write stop record #X

Data to follow size word: Stop record (variable length as defined by user)
Good response: Command 6 - Ack
Error response: Command 9 - Error Message

Note: No response if this command was issued in response to a Read Stop command.

This command will be followed in the message buffer by a valid route stop record as defined in Appendix 1 of this document. The stop will be inserted into the TCP-CO's route information at position X where x is 0-15.

Command 48 + X - Delete stop record #X

Data to follow size word: None
Good response: Command 6 - Ack
Error response: Command 9 - Error Message

This command instructs the receiving unit to remove the stop at position X from the route information, where x is 0-15.

Sample Commands

Sample of individual database element READ command transmission (fits in single packet):

STX	Dest	Src	Sq.#	Size	Command	Sz-Hi	Sz-Lo	Register ID	checksum
0x02	0x01	0x15	0x00	0x07	0x01	0x00	0x04	0x06 0x02 0x01 0x01	CRC32

Note: first (packet) size field is packet size not including CRC32, second size word is command argument size.

Sample of individual database element Write command: (also single packet)

STX	Dest	Src	Sq.#	Size	Command	Sz-Hi	Sz-Lo	Register ID	Data	chksm
0x02	0x01	0x15	0x00	0x08	0x01	0x00	0x05	0x06 0x02 0x01 0x01	0xNN	CRC32

Note: data field format determined by data type - first byte in register ID specifies type

Sample Read Stop #1 command:

STX	Dest	Src	Sq.#	Size	Command	Sz-Hi	Sz-Lo	chksm
0x02	0x01	0x15	0x00	0x03	0x01	0x00	0x00	CRC32

Sample Delete Stop #3 command:

STX	Dest	Src	Sq.#	Size	Command	Sz-Hi	Sz-Lo	chksm
0x02	0x01	0x15	0x00	0x03	0x01	0x00	0x00	CRC32

Note: the commands for stop interaction are as follows:

0x10 plus the zero-based array index to read a stop
0x20 plus the array index to write a stop
0x30 plus the array index to delete a stop

Appendix 1: Route Stop Record Format

Field	Label	Data Type	Max Length {Range}
1	Truck ID	Alpha	20
2	Transaction Number	Numeric	12
3	Lease ID	Alpha	20
4	Tank ID	Alpha	20
5	Recommended LACT ID	Alpha	20
6	Load Date	Date	10
7	Seal Open Time	Time	8
8	Seal Closed Time	Time	8
9	Load Average Temp	Numeric	8
10	Load Avg Dens	Numeric	8
11	Load Avg BS&W	Numeric	8
12	Volume - Indicated	Numeric	12
13	Volume - GRS	Numeric	12
14	Volume - GST	Numeric	12
15	Volume - NSV	Numeric	12
16	Mass	Numeric	12
17	Unauthorized NSV	Numeric	12
18	CTL	Numeric	12
19	CSW	Numeric	12
20	CCF	Numeric	12
21-40	Prompt #1 - #20 Response	Alpha	User Defined
41	Report To Use	Numeric	2 {01-16}
42	Comment Field	Alpha	Variable
42+n	User Defined Field n	Alpha	Variable

Note: User-Defined Fields are any TCP-RAD database record fields, fixed or user-defined, NOT ALREADY INCLUDED IN THE STOP DATA RECORD that are selected by the dispatcher to be downloaded as part of the stop definition. There will not be a limit to the number of entries, but the overall size will limited to the maximum size of a stop record (1K).

The stop record will be transmitted as a series of null-delimited strings. Leading and trailing whitespace in each field will not be transmitted, but whitespace is permissible within each field otherwise. Unused or empty slots (or whitespace-only) in the stop record will still contain a placeholder null. For example, an empty field would be represented as follows (prevField is the data in the field prior to the empty field and nextField is the data in the field after the empty field) :

```
...<NULL>prevField<NULL><NULL>nextField<NULL>...
```

Appendix 2: Recommended Application Layer Interface

The following is an example implementation that describes the division in functionality in the protocol layer and the application layer. It is intended merely as an example and is not necessarily representative of the final implementation.

The application will interact with the communications stack via a well-defined interface providing access to all of the required services provided by the communications protocol. The interface will at a minimum provide the following functions to the application layer:

Initialize Communications Stack (select protocol, various communications parameters)
Send Message via current packet protocol (transmit command)
Get Message via current packet protocol (receive command)
Status of Communications Stack (Retrieve status, error information, error text)

The actual interface may be enhanced or hidden by higher level functions on each platform, but at minimum the above requirements will be implemented as functions that will accept the following arguments and return the specified information:

Initialize Communications Stack

```
int InitComm(int protocol, COM_OPTIONS pOptions);
```

arguments:

protocol -	integer indicating the selected protocol in the available list. For initial release of TCP-CO, the available protocols will be 0-none, 1-Direct Connect and 2-Radio Packet Protocol, different only in maximum size of the data field in the packet (248 for direct connect, 32 for radio).
pOptions -	structure with the following information: port to use, baud, data, and parity settings, local address, and other information required to define the communications setup.

Return value:
0 on success, nonzero integer indicating the error on error. Error information may be retrieved by the ComErrMsg() function implementation.

Send Message via current protocol

```
int SendMessage(char *buf, unsigned int msgSize, int destination_address);
```

arguments:

buf -	character buffer containing message to be sent
msgSize -	Unsigned integer value containing number of characters to send
destAddr -	integer specifying destination for this transmission.

Return value:
0 on success, nonzero integer indicating the error on error. Error information may be retrieved by the ComErrMsg() function implementation.

Get Message via current protocol

unsigned int GetMessage(char *buf, int maxSize, unsigned int waitTime);

arguments:

buf -	character buffer to hold response
maxSize -	Unsigned integer value containing maximum number of chars to write before returning to avoid exceeding size of *buf*
waitTime -	If zero, return immediately with current status. If nonzero, wait waitTime milliseconds or until a completed message is available before returning.

Return value:
0-no complete message available
1-65534 - size of successfully returned message
65535 - Overrun or other error - use GetComError() for more information

Status of Communications Stack

int GetComError(void);

arguments:
none

Return value:
0-no error present
nonzero - indicates an error

char *ComErrMsg(int errorID);

arguments:

errorID -	Value returned by GetComError. If -1, returns message associated with the last known status of the communications stack.

Return value:
pointer to a character message describing the error associated with *errorID*